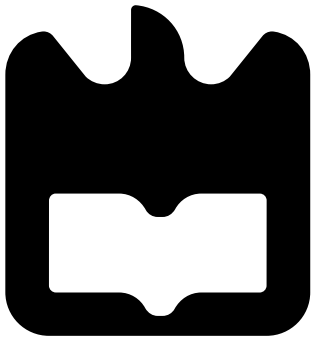
Universidade de Aveiro Departamento de Engenharia Mecânica
2015

Ivan
dos Santos Ferreira

Integration of the planning, global localization and path execution in autonomous cars

Integração do planeamento, localização global e execução de caminhos na navegação de automóveis autónomos



Ivan
dos Santos Ferreira

Integration of the planning, global localization and path execution in autonomous cars

Integração do planeamento, localização global e execução de caminhos na navegação de automóveis autónomos

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requesitos necessários à obtenção do grau de Mestre em Engenharia de Automação Industrial, realizada sob a orientação científica de Vítor Manuel Ferreira dos Santos, Professor Associado do Departamento de Engenharia Mecânica da Universidade de Aveiro e de Manuel Bernardo Salvador Cunha, Professor Auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro

o júri / the jury

presidente / president

Prof. Doutor Pedro Nicolau Faria da Fonseca
Professor Auxiliar da Universidade de Aveiro

vogais / examiners committee

Prof. Doutor Carlos Fernando Couceiro de Sousa Neves
Professor Coordenador do Instituto Politécnico de Leiria - Escola Superior de Tecnologia e Gestão

Prof. Doutor Vítor Manuel Ferreira dos Santos
Professor Associado da Universidade de Aveiro (Orientador)

agradecimentos / acknowledgements

É com muito gosto que aproveito esta oportunidade para agradecer ao Prof. Doutor Vítor Manuel Ferreira dos Santos pela disponibilidade e apoio, pelo conhecimento transmitido e por me motivar ao longo deste tempo.

Queria também agradecer ao Prof. Doutor Manuel Bernardo Salvador Cunha pela sua disponibilidade e conselhos.

Agradeço também a todos os membros do laboratório que me ajudaram durante este tempo e pela boa disposição que traziam para o laboratório. Um especial obrigado ao Jorge Almeida pelo apoio, paciência e conhecimento transmitido.

Aos amigos e colegas de curso pela boa disposição, companheirismo, pela ajuda em tempos mais difíceis, e pelo enorme gosto que tem por estas matérias.

Aos meus pais um especial agradecimento pelo enorme apoio que me tem dado ao longo dos tempos, pelo esforço e pela dedicação que tiveram para que eu conseguisse concretizar os meus objectivos. Sem vocês nada disto seria possível.

Palavras-chave Navegação, Planeamento, GPS, ROS, C++, PHP, JavaScript, Html, SQL

Resumo

O Atlascar é um projeto desenvolvido no laboratório de automação e robótica, no departamento de Engenharia Mecânica da Universidade de Aveiro. Este veículo é utilizado para investigação em condução autónoma e sistemas avançados de apoio ao condutor.

O objetivo desta dissertação é desenvolver um sistema de navegação para planeamento de rotas. Este sistema inclui uma interface gráfica que permite ao utilizador selecionar um destino, com pontos de passagem intermédios se assim for pretendido. A aplicação deve permitir monitorizar em tempo real o veículo e indicar as manobras que devem ser realizadas ao longo do percurso. Novo *hardware* terá de ser instalado e configurado, nomeadamente um novo servidor e recetor GPS. Isto requer uma intervenção na configuração atual do Atlascar.

O sistema está dividido em três partes: Planeador, Navegador e Interface Gráfica. O módulo do planeador é responsável por calcular os percurso e as manobras a realizar. O navegador é responsável pelo processamento dos dados vindos do planeador. Estabelece uma comunicação com uma base de dados de onde vai ler e escrever valores. Este módulo também sera responsável por ler os valores vindos do GPS. A interface gráfica permite ao utilizador selecionar o destino pretendido, o que é feito clicando num mapa. Este módulo também permite monitorizar a localização do veículo, ver o percurso que irá ser realizado e apresenta a instrução a realizar. Este módulo também estabelece uma comunicação com uma base de dados. Este módulo é opcional porque o sistema também funciona sem uma interface visual, esta simplesmente permite ao utilizador fazê-lo de forma interactiva e também permite monitorização. Caso o utilizador escolha um percurso errado durante a viagem, o sistema recalcula a missão e apresenta um novo caminho.

Keywords Navigation, Planning, GPS, ROS, C++, PHP, JavaScript, Html, SQL

Abstract

Atlascar is a project developed in the Automation and Robotics laboratory at the Mechanical Department of the University of Aveiro, Portugal. It is used for research in autonomous navigation and for advanced driver assistance systems.

The aim of this thesis is to develop a navigation system for path planning, which includes a graphical interface that allows the user to input a final destination, with possible via points if needed. The application should be able to track the vehicle in real-time and output sequences of high level maneuvers along the mission. New hardware is also to be installed and configured, namely a new server and a new GPS receiver, which requires an intervention on the previous setup of the Atlascar.

The whole system is divided into three parts: Planner, Navigator and User interface. The planner module will be responsible for calculating the path and the instructions. The navigator module will be responsible for processing the data from the planner module ; communicating with a database, read and write values and acquiring data from the GPS. The user interface allows the user to input the desired destination by clicking on a map, this module also allows the user to: monitor the current location of the vehicle; view the path to the destination and it shows the instruction that needs to be done. This module also establishes a communication with the database. This module is optional because the system also works without a user interface, it simply allows the user to do it interactively and it allows motorization. If a wrong decision is made and the user follows a wrong path, the system will recalculate the mission and show a new path.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Navigation has been used for a long time, for example by using maps and a compass. We "navigate" ourselves everyday to our destination. During the navigation process it might happen that we enter an unknown location and we probably might get lost. What now ? Well, this is where the commercial GPS (Global positioning system) navigation devices come into action. They help us getting from point A to point B without getting lost. Nowadays, we use commercial navigation solutions on our smartphones and cars to plan our journey or to get more information about a specific place. If we add to the GPS technology a digital map, we can create a precise navigation solution. For example, in our car, we use the navigation system to set up a destination on the map and it will generate a route so that we can drive there based on its instructions.

Vehicles play an important role in modern society. But, as society evolves and grows, more and more vehicles enter our roads, which generates more traffic and consequently more accidents. This could be reduced by a more organized traffic, or in a near future, by intelligent autonomous vehicles. Times are changing and with the evolution of autonomous cars, in a few years (about year 2020) [Baptiste(2015)] we will simply enter in our car, tell it where we want to go and the only thing we will need to do is to sit back and enjoy the ride. This could reduce the accidents because the vehicles will possibly communicate with each other using the vehicle-to-vehicle technology. This type of communication allows cars to broadcast their position, speed, steering-wheel position, brake status, and other data to other vehicles within a few hundred meters. The other cars can use such information to build a detailed picture of what's unfolding around them, revealing trouble that even the most careful and alert driver, or the best sensor system, would miss or fail to anticipate [Knight(2015)].

1.1 Context and Motivation

Atlascar, figure 1.1, is a project developed in the Automation and Robotics division at the Mechanical Department of University of Aveiro, Portugal. Project Atlascar 1 is a real car (Ford Escort SW) that is used for research in autonomous navigation and ADAS (Advanced driver assistance systems). The general philosophy has been to enrich the car with different kinds of sensors, to account for different types of perception and cover for redundancy, and thus allowing different types of research in data fusion and interpretation for the future developments in this project. The purpose of the vehicle transcends the mere massive collection of data, which it can do very well, but also uses that data to create models for enhanced percep-

Figure 1.1: Atlascar

tion and data fusion [Santos and Almeida(2010)]. It is equipped with several state of the art hardware. Much software has already been developed, but it is missing a navigation software for path and mission planning. We can split the developments in navigation into two parts: Local navigation and Global navigation. Local navigation is related to obstacle detection and avoidance, lane detection, target tracking, maneuvers, among others. In this area several solutions have already been developed. On the other hand, we have Global navigation, which is related to mission planning and localization. Here, little developments have been made in the context of the Atlascar project, and there is the need for an infrastructure to merge the local and global navigation components. Atlascar already features technologies that allows it to be partially autonomous. It currently features technologies previously developed and some examples are: road detection [Morais(2014)], automatic gearbox [Pinho(2014)] and throttle [Ramalhinho(2011)], automatic parking [Pereira(2012)] and pedestrian detection [Silva(2013)] [Azevedo(2014)] among many others. The work developed in this dissertation will contribute to get one step closer to the final goal, which is a fully autonomous car. There is also the need to reorganize the components, cables and its labels in the car because during these years of modifications it has become very disorganized.

1.2 Objectives

The main objectives of this work are:

- Installation of the new hardware for the Atlascar, namely: a more powerful server and a high precision GPS;
- Development of a graphical interface that allows choosing the start and destination coordinates in an interactive way; the application should also allow the user to monitor in real-time the steps of the mission;

- Development of a module to execute the mission planning that uses high level maneuvers or decisions that need to be executed during the mission (e.g. follow road, change direction, turn in the next crossing, turn around, stop/park, etc.);
- In a more advanced phase, this module should be able to select and execute the necessary maneuvers.

1.3 Structure of the document

This thesis is divided into six chapters and the appendix section.

Chapter One This chapter presents an introduction to the problem and explains what already exists and what needs to be done;

Chapter Two This chapter presents the state of the art where technologies related to autonomous cars are presented. Definitions related to localization and navigation are also explained;

Chapter Three This chapter presents the current infrastructure and the modifications done on the car. The software infrastructure that will be used is also explained. The new hardware is also shown;

Chapter Four This chapter presents the solution found to solve the problem and explains all the software developed in detail;

Chapter Five This chapter presents the experiments made and the obtained results will be analysed;

Chapter Six This chapter presents the conclusions and the future work that needs to be done.

Appendix The appendix contains all the instructions to install and use the software. Contains explanations on how to execute the developed system.

Chapter 2

State of the art

The main topics that are relevant for this thesis are localization, planning and routing. This chapter will focus on these topics.

2.1 Autonomous Cars

Autonomous Cars is a topic that is getting more and more popular. Numerous researches have been made in this area by universities, companies and vehicle manufacturers. Research on this field started long time ago. On figure 2.1 we can find the evolution of what has been done, and what will possibly happen in this field.

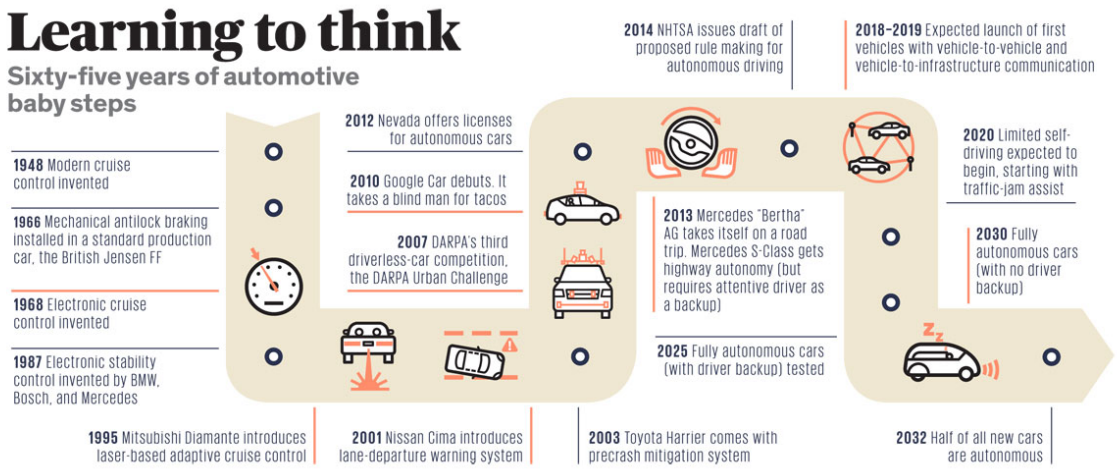


Figure 2.1: Autonomous Cars Timeline [IEEE(2012)]

In a few years we will see fully autonomous cars on the roads but for now there are still several issues related to security (where 99% is not enough) and navigation that need to be solved. There are already some vehicles that drive autonomously, for example the cars from Google, but its not yet a commercial solution. Nowadays, there are already semi-autonomous cars on the roads and the main technologies are [McBride(2008)]:

- Anti-lock brake systems(ABS), imminent collision warning;

- Cruise control, urban cruise control(UCC) able to recognize stop signs and traffic signals);
- Lane departure warning (LDW), lane keeping assistance (LKA);
- Traction control, electronic stability control (ESC), active suspension;
- Pre-crash sensing with occupant protection equipment (airbags, seat belts);
- Blind spot detection, pedestrian detection, parking assistance, night vision;
- Driver drowsiness and distraction monitoring;
- Vehicle to vehicle and infrastructure integration (VII / V-V), intelligent vehicle highway systems (IVHS);
- Total accident avoidance / autonomous vehicle control;

More features are coming soon and according to [James(2013)] the timeline for autonomous cars and security features are the following:

2015

- "Super cruise":autonomous steering, braking and lane guidance at speed;
- Autonomous throttle, gear shifting, and unoccupied self-parking;

2018

- Release of Google's autonomous car technology.

2020

- Volvo expects accident-free cars and GM, Audi, Nissan and BMW expect fully autonomous driverless cars.

It is expected that in 2040 almost 75% of the cars will drive autonomously [IEEE(2012)].

With the implementation of autonomous cars, we could possibly reduce accidents, improve productivity because we could work while we drive and have more organized traffic.

2.1.1 GNSS

GNSS (Global Navigation Satellite System) is used to describe the collection of satellite positioning systems that are now operating or planned [Jeffrey(2010)].

GPS (United States)
GPS was the first GNSS system and, as of today, it is the only one that is fully operational. GPS was launched in the late 1970's by the United States Department of Defense. It now uses a constellation of between 24 and 32 satellites, and provides global coverage.

GLONASS (Russia)
GLONASS is operated by the Russian government. The full GLONASS constellation will consist of 24 satellites. Global coverage is provided since 2011.

Galileo (European Union)
Thirty satellites are planned with the first being launched in 2006. The full constellation will not be complete for several years.

Compass (China)
Compass will be the Chinese navigation satellite system. The system will consist of 35 satellites. A regional service is provided since 2011 then the service will be extended to provide global coverage in the years 2015-2020.

A more detailed explanation on how GNSS works can be found in: [Jeffrey(2010)].

2.1.2 Localization

Localization provides estimates of the location, attitude, velocity and acceleration of the vehicle with respect to some fixed coordinate system.

Geographic Coordinate System
This is the type of coordinates that is used in this thesis and consists of using measures of latitude and longitude to determine location. The earth globe is divided into lines as represented on figure 2.2.

The ones belonging to the latitude are parallel to the equator line. Positive numbers represent the northern hemisphere, negative numbers represent the southern hemisphere and the equator represents 0 degrees of latitude, while both poles represent 90 degrees. All other points range between 0° to 90° north or 0° to 90° south. Lines of longitude run east and west parallel to the prime meridian. Positive numbers represent the eastern hemisphere, negative numbers represent the western hemisphere, and the prime meridian represents 0 degrees of longitude. All other points range between 0° to 180° east and 0° to 180° west. Lines of longitude are not parallel. The closer they are to the poles, the shorter the distance between them [NIIMS(2007)].

According to [NIIMS(2007)] there are three primary ways of describing locations using latitude and longitude coordinates:

1. Degrees Minutes Seconds (ddd° mm' ss.s")
   This is the most common format that is used on maps. (e.g. 43° 23' 45" , 71° 8'36")

2. Degrees Decimal Minutes (ddd° mm.mmmm')
   This format is used by aircraft guidance systems: (e.g. 43° 23.75' Latitude , 71° 8.6' Longitude)

3. Decimal Degrees (ddd.dddd°)
   This is used by the American National Weather Service as well as based mapping systems. This is the type of coordinates that will be used. (e.g. 43.395833° Latitude, 71.143333° Longitude)

GPS
A common GPS is capable of accuracies of the order of 10m. Differential GPS is capable of accuracies of better than 0.5m, and real-time kinematic GPS is capable of accuracies of around 2cm. GPS systems can fail in a number of different ways. Most common failure modes involve obstruction of line-of-site to satellites, multipath from foliage or terrain geometry, and

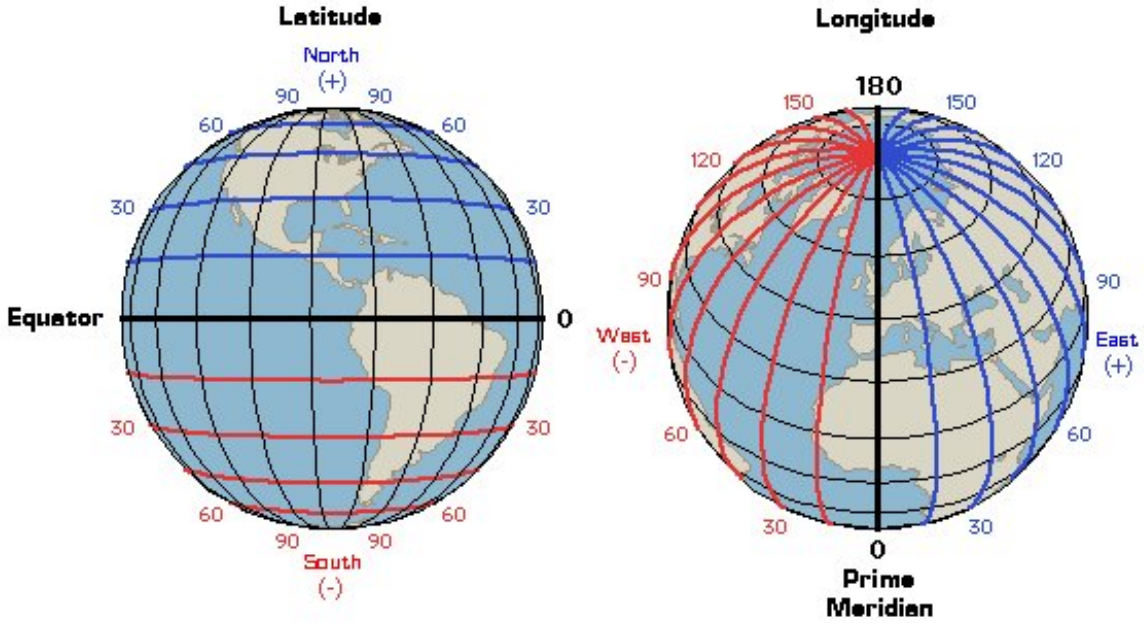


Figure 2.2: Latitude-Longitude Map [Geographyworldonline(2008)]

active jamming from other RF (Radio Frequency) sources [Durrant-whyte(2001)].

INS

An INS (Inertial Navigation System) uses information from an IMU (Inertial Movement Unit) to compute accurate position over time. Inertial sensors have a major advantage in being non-radiating, non-jammable sensors which do not rely on any external information to provide estimates of position, attitude and body rates. An IMU has enough precision to allow vehicles to run without any external navigation updates for several minutes. This is often enough to overcome intermittent jamming of GPS or slow acquisition of other landmark data (for example). The use of INS in Autonomous Vehicle applications is now a standard practice [Durrant-whyte(2001)].

Novatel's SPAN-IGM-A1 unit which will be used in this work, combines GPS with INS to create a precise navigation solution. An example is illustrated in figure 2.3. We can observe that if we would only use a GNSS solution and drive for example trough tall buildings or inside a tunnel we would eventually lose signal. In addition using only an INS is not enough because after a while it is required to adjust the output. So, by combining both technologies we obtain a precise navigation system that overcomes the limitations of the individual technologies.

Two examples of GPS/ INSS integration architectures that can be implemented are [Bevly and Cobb(2010)]:

Loose Coupling

A loosely coupled GPS/INS integration routine combines the inertial measurements with position and velocity measurements calculated by the GPS receiver. Feedback to the inertial processor calibrates the IMU to remove effects from biases, scale factors, and/or misalignment.

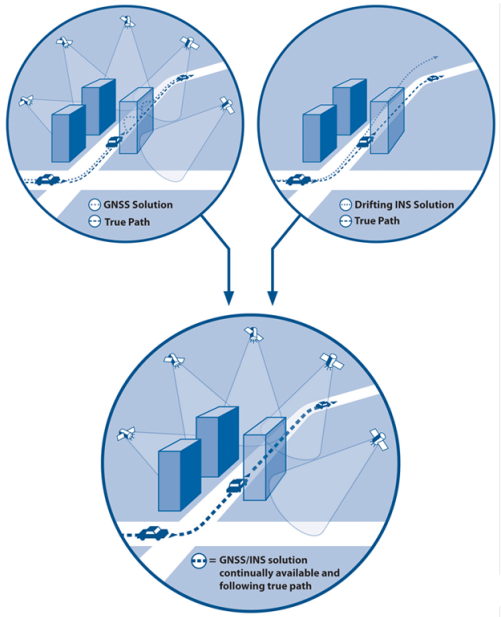


Figure 2.3: Combined GNSS/INS [Novatel(2014a)]

Close Coupling

A closely coupled GPS/INS integration routine combines the inertial measurements with range information to the GPS satellites provided by the GPS receiver. Feedback to the inertial processor is retained, but there is no feedback to the GPS receiver. Range information includes a user to satellite distance and user to satellite velocity.

2.2 Navigation

There are five basic forms of navigation [Andrews(2007)]:

1. *Pilotage*, which essentially relies on recognizing landmarks to know where we are and how we are oriented.

2. *Dead reckoning*, which relies on knowing where we started from, plus some form of heading information and some estimate of speed.

3. *Celestial navigation*, using time and the angles between local vertical and known celestial objects (e.g., sun, moon, planets, stars) to estimate orientation, latitude, and longitude.

4. *Radio navigation*,which relies on radiofrequency sources with known locations (including global navigation satellite systems satellites).

5. *Inertial navigation*,which relies on knowing your initial position, velocity, and attitude and thereafter measuring your attitude rates and accelerations. It is the only form of navigation that does not rely on external references.

Navigation is concerned with the acquisition of, and response to, external sensed information. The navigation function takes input from sensors observing the operational environment.

It must use this information to create an internal representation of the environment that can subsequently be used in the execution of a mission.

2.3 Routing

Mission and task planning functionally generates trajectories, behaviours or way points for the system as a whole. It has no direct links with either sensory input or controller output. However, it clearly must use an understanding of these, in conjunction with prior maps and defined mission objectives, to produce appropriate navigation commands [Durrant-whyte(2001)].

There are differences between Route Planning and Path Planning [Lopez(2015)]:

Route planning This topic focuses on determining the route between two geographical locations, or how the vehicle will traverse from its present location to a destination, which could be very far away.

Path planning Also known as Motion planning. This topic discusses the creation of a series of path segments that describe the path directly in front of (or behind) the vehicle.

There are a lot of map providers, which provide a web interface to calculate a route between two points (with intermediate points, if desired). Some of them also provide an API (Application Programming Interface) that can be used to develop specific mapping software based on the API (or APIs) we choose. Some of the most common providers are shown in table 2.1. Figure 2.4, 2.5, 2.6 and 2.7 show the website from a few providers.

Table 2.1: Routing Software comparison

Provider	API	Open Source
Graphhopper	Yes	Yes
OpenStreetMap	Yes (3rd Party)	Yes
Google Maps	Yes	No
MapQuest	Yes	No
Mapbox	Yes	No
Nokia Here	Yes	No
Yahoo! Maps	Yes	No
Bing Maps	Yes	No
ViaMichelin	Yes	No

There is also software that allows to work offline (e.g. OpenSourceRoutingMachine, pgRouting, OpenTripPlaner), which has advantages and disadvantages. We can benefit from these solutions in case we don't have internet access in the car or if we frequently drive through areas where the internet access is poor. We also don't rely on the website that might be unavailable when we need it. But, of course, there are also disadvantages; for example, the speed of the service depends on the hardware and it is necessary to manually download the maps, to perform an update for the maps. For example the data size for a map of the whole planet is about 29GB, Europe about 16GB and Portugal, only 108MB. The maps can be obtained from Geofabrik [Geofabrik(2015)] and OpenStreetMaps [OpenStreetMap(2015)] .

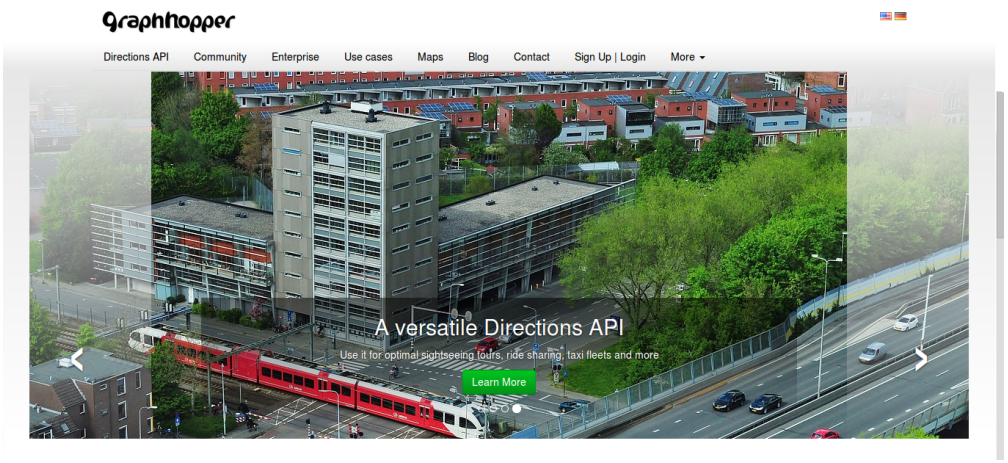


Figure 2.4: Routing software provided by Graphhopper [GraphHopper(2015)]

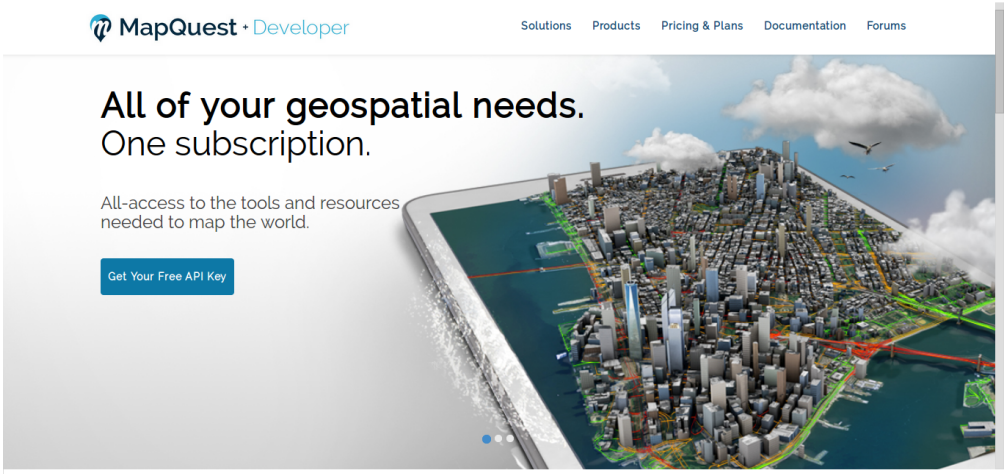


Figure 2.5: Routing software provided by Mapquest [MapQuest(2015)]

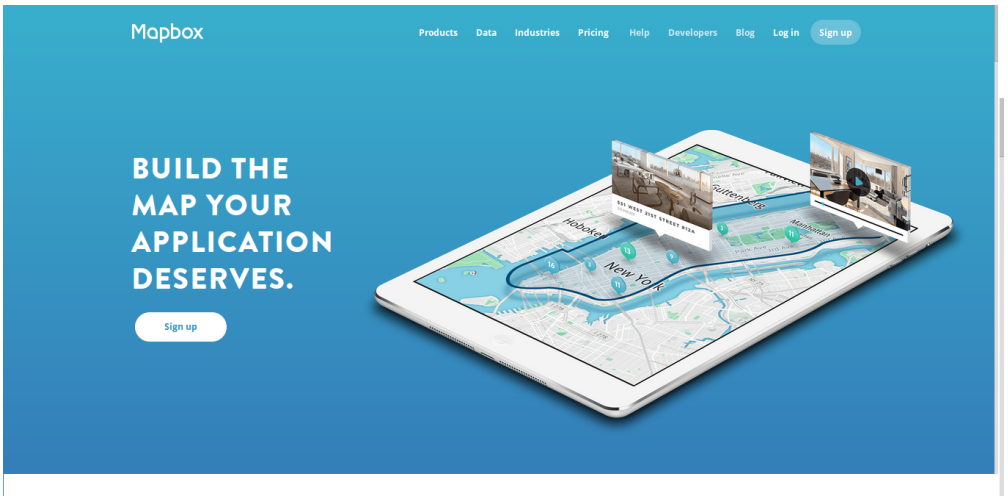


Figure 2.6: Routing software provided by Mapbox [Mapbox(2015)]

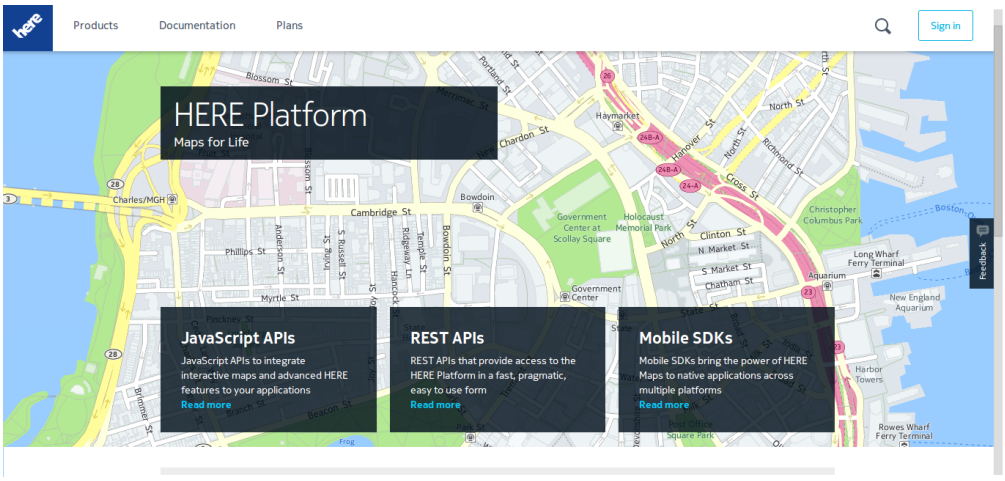


Figure 2.7: Routing software provided by Nokia Here [Nokia(2015)]

OpenSourceRoutingMachine (OSRM) This project can be split into two parts: Server(backend) and Webpage(frontend). It is written in C++, very fast and consumes a low amount of resources. The map data originates from the OpenStreetMap Project.

pgRouting pgRouting is an extension to the PostGIS/PostgreSQL geospatial database and adds routing functionality. It is mainly based on SQL (Structured Query Language) and no turn restrictions are available.

OpenTripPlaner (OTP) OpenTripPlaner runs on a Java web server. It bundles a REST Api and a WebFrontend. Needs more resources than OSRM and is also slower in calculating routes.

Comparing these three services, OSRM was chosen because it fulfils the needs of this project, very fast, the maps can easily be updated, it works offline and is written in C++.

Chapter 3

Experimental Infrastructure

This chapter describes the car and the related interventions, the new equipment that has been installed and some preliminary tests that have been made.

3.1 Atlascar

Currently the vehicle is equipped with several equipment and software that allows performing different tasks related to autonomous driving. Some equipment is shown in figure 3.1. As mentioned in chapter 1 some examples of the existing technologies are:

- road detection [Morais(2014)]
- target detection with laser [Almeida(2010)]
- pedestrian detection [Silva(2013)], [Azevedo(2014)]
- driver monitoring [Oliveira(2012)]
- vehicle monitoring and remote control [Rocha(2011)]
- automatic gearbox [Pinho(2014)]
- automatic throttle [Ramalhinho(2011)]
- automatic parking [Pereira(2012)]

The equipment aboard the Atlascar can be divided into four areas, namely:

Power generation
The Atlascar has a more powerful alternator to provide more alternate current, an inverter to change the direct current (DC) to alternate current (AC), a secondary battery with a power and current indicator to store the energy, an Uninterruptible Power Supply and a power distribution panelboard that contains two power sources (24V and 12V) and a PLC (Programmable Logic Controller) for turning the sensors on and off.

Perception/vision
Atlascar is equipped with a 3D camera, stereo camera, two planar 2D laser sensors, foveal vision system and a 3D laser.

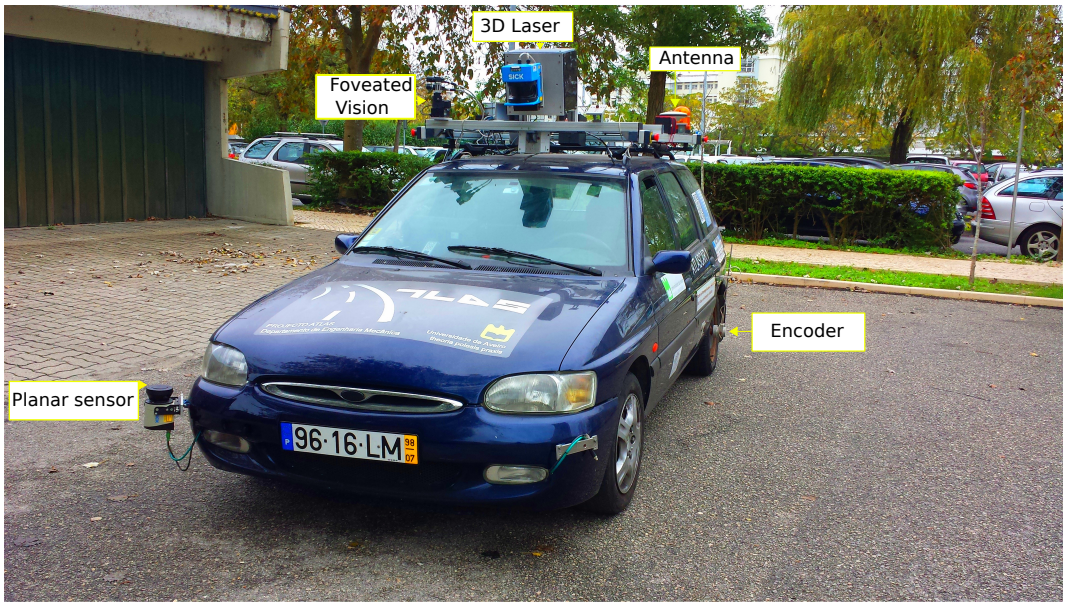


Figure 3.1: Some equipment featured in the Atlascar

Sensors and accessory tools

Atlascar has two PLC for turning the sensors on and off, an inertial movement unit, a GPS, an encoder to measure the speed, a unit that allows remote monitoring and several selectors that allow commuting between manual and automatic mode for brakes, steering, clutch, gearbox, handbrake and engine.

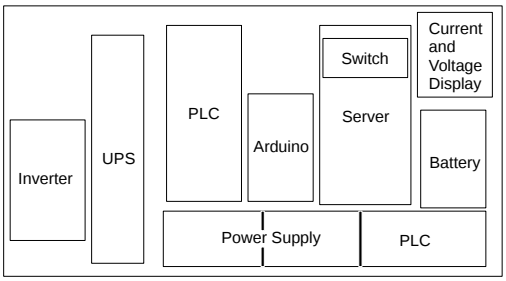
Data processing and management

The Atlascar has a powerful server and two network switches.

3.2 Atlascar Hardware

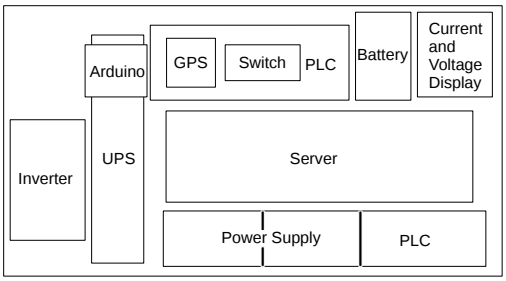
An intervention on the hardware level was required because new hardware had to be installed, namely a new Server and the new GPS along with its antenna. The new server was necessary because the car required more processing power. In addition, the new GPS was required for this thesis because it is much more accurate and has more features than the old unit, as it will be explained later. This new GPS is very important for this work because in the future, when the car hopefully drives by itself, it will need a reliable equipment that is very accurate and that works better in places where the old unit might not perform so well. However, as the new server is much bigger, the old trunk layout was not appropriate, figure 3.2a and 3.2b. A new layout, figure 3.2c and 3.2d, was designed so that all components would fit correctly and to guarantee that the server has enough space to get a good air flow. The PLC board had to be moved and a new place for the battery, Switch and the new GPS receiver had to be found. It was also ensured that all components are not loose and don't collide with each other during travel, therefore, supports made of styrofoam where placed between the components.

A new support has been installed on the roof of the car to hold the external GPS antenna, figure 3.3, therefore an old support already installed on the car was moved to a new place

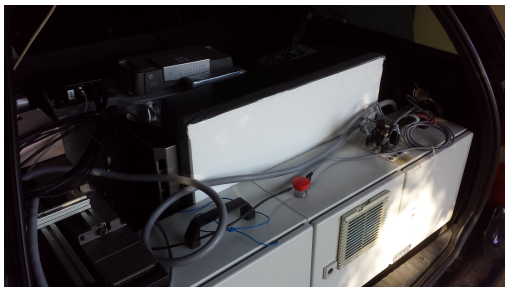


(a) Old Schematic

(b) Old Picture



(c) New Schematic



(d) New Picture

Figure 3.2: Atlascar Trunk Layout Schematics and Pictures

and adapted so that it can properly hold the antenna.

An old case was removed and replaced by a new one. This was necessary because the old case was too big and heavy, and in its interior there was only an Arduino, figure 3.4a and 3.4b. The new case is much smaller and lighter, figure 3.5. Some unnecessary cables where also removed from the car, and a few cables that had no identification got one.

The throttle was switched from automatic mode, figure 3.6a to manual, figure 3.6b. This is done by opening the case identified as number "1" and removing the cable. The cable which comes from the motor controlled by the arduino, identified as number "3" and connects to number "2" is also removed. To change to manual acceleration the cable that was removed from number "1" is placed on the location marked as "2".

The new server that has been installed is much heavier than the previous one and this causes the chassis to hit the sidewalk when trying to park it next to the Mechanical Department. So two ramps, figure 3.7, where made in order to prevent this from happening.

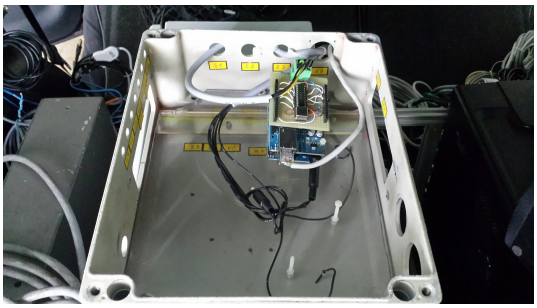
3.2.1 GNSS

In order to provide a highly accurate localization, a new GNSS unit was installed. The equipment used is the Novatel SPAN-IGM-A1 unit, figure 3.8a. It combines a high precision Global Navigation Satellite System (GNSS) with an Inertial Measurement Unit (IMU). It can be configured to work as a GNSS+INS engine or as a standalone IMU sensor. This unit will be responsible for determining the vehicle localization. Table 3.1 compares the old unit, figure 3.8b with the new one, figure 3.8a.

To establish a communication with this device was not a trivial task. Not only because this device doesn't work as a plug-and-play unit, but also because it needs to be configured

Figure 3.3: GPS Antenna and Support

(a) Closed



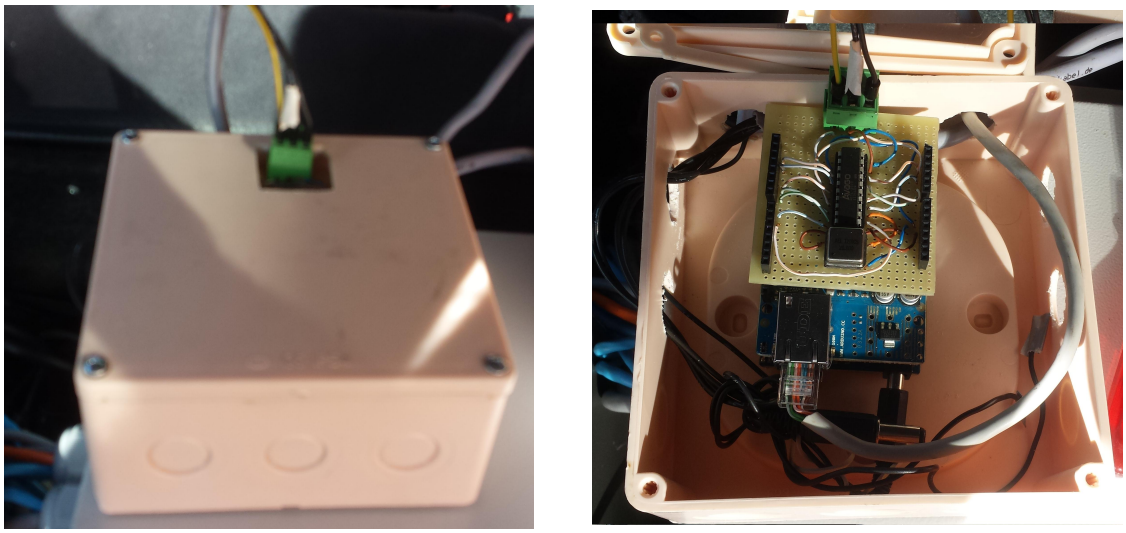
(b) Opened

Figure 3.4: Old box for the odometry control unit

---

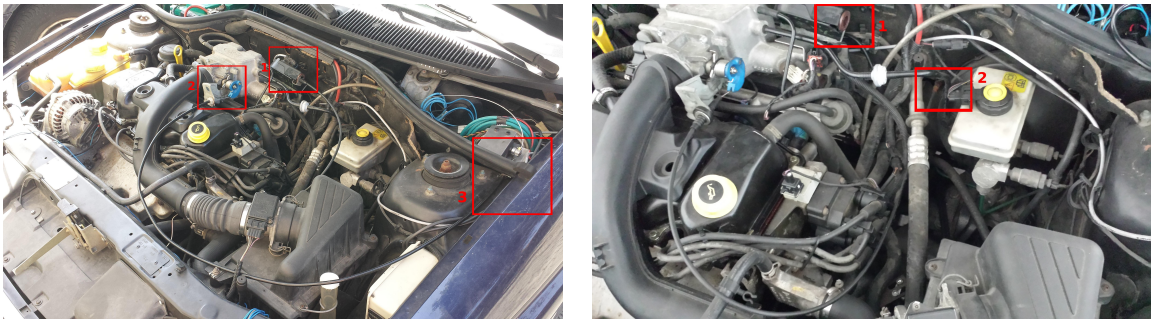
before using it. Finding a good ROS (Robot Operating System) package for this unit required some time because there are very few packages for this device. And the ones that exist are unclear to install and don't work as expected. But at the end the *novatel* package from GAVLab was successfully installed. Even with this package installed the communication did not work. This had to do with the USB (Universal Serial Bus) connection that wasn't being recognized by the operating system. The solution was found on the Novatel website [Novatel(2015)], the procedures to fix this issue are described on Appendix A.4. Finally it was possible to communicate with the GPS. To test the precision of the new equipment, it was placed outside through a window of the Automation and Robotics laboratory 3.9 and using ROS, the coordinates where logged during ten hours into a ROS bag. Figure 3.10 and 3.11 show the Latitude and Longitude values that were recorded during the ten hours.

With all the data processed a standard deviation was calculated, the result was 0.00001406. Accordingly to [Osborne(2013)] the radius of the equator is: 6,371,008 meters. So the circumference of the earth will be 40,030,173 meters. The equator is divided into 360 degrees of latitude, so dividing the circumference of the earth by 360 degrees of latitude the result will



(a) Closed (b) Opened

Figure 3.5: New box for the odometry control unit



(a) Automatic (b) Manual

Figure 3.6: Throttle modes

be that each degree of latitude corresponds to 111,195 km. Table 3.2 illustrates the precision based on the decimal places. So with the standard deviation that was obtained the precision that could be obtained was 1.57 meters. This precision was achieved between two buildings and not on an open field. The precision on an open field would possibly be better because the signals from the satellites would not be obstructed by buildings.

3.2.2 Server

This new server will be responsible for processing all the information that comes from multiple sensors and to seamlessly run the required software. This new server is more powerful than the older one, and a comparison between the old server, figure 3.12b, and the new server, figure 3.12a, can be found in table 3.3 [Hewlett Packard(2015)].

Since the server was completely new, Ubuntu 14.04 was installed, the repository of the laboratory (LarTkV5) was set up and all software required for this thesis was installed.

Figure 3.7: Ramp that helps to avoid the collision of the atlascar's chassis with the sidewalk

Table 3.1: Comparison of the new GPS that will be installed and the old unit that will be removed

	Novatel-SPAN-IGM-A1	GPS-353-BU
RMS	Single point L1/L2 - 1.2m, DGPS - 0.4m	5m WAAS enabled, 10m WAAS disabled
Data Rates	GNSS measurement & position - 20Hz, IMU measurement & solution 200Hz	1Hz
IMU	Yes	No
COM Ports	1xUSB, 1xRS-232, 1xCAN Port	1xUSB

(a) Novatel-SPAN-IGM-A1 [Novatel(2014b)]

(b) GPS-353-BU [USGlobalSat Inc.(2014)]

Figure 3.8: New and old GPS units

Figure 3.9: Test conditions of the new GPS Antenna

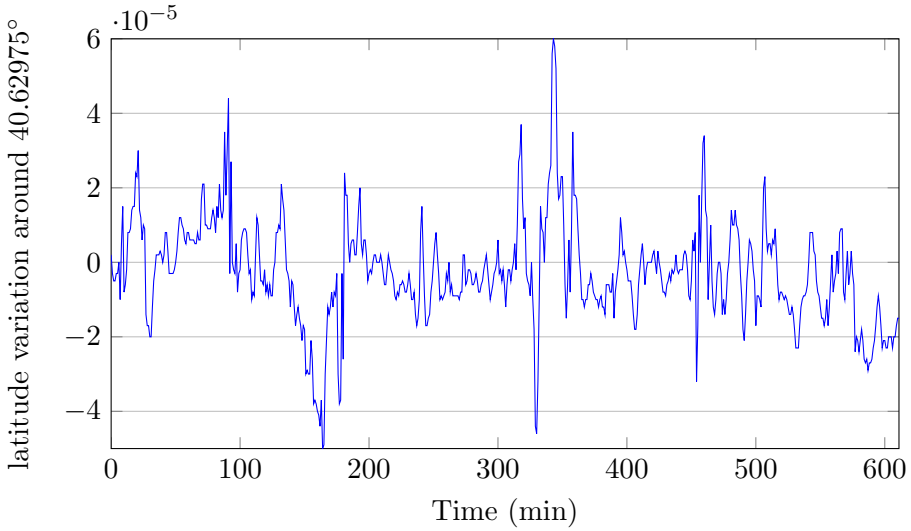


Figure 3.10: Latitude Graph of the GPS test measures

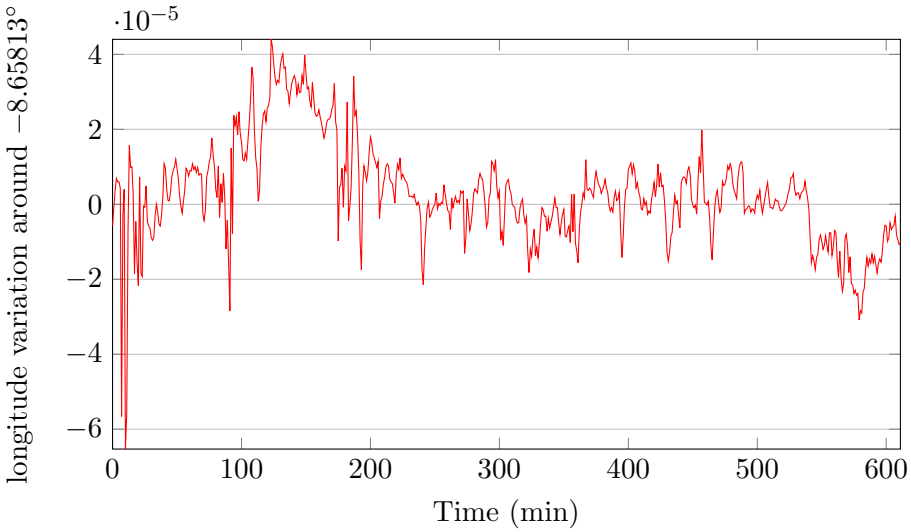


Figure 3.11: Longitude Graph of the GPS test measures

Table 3.2: Decimal degrees precision

decimal places	decimal degrees	distance
0	1	111.92 km
1	0.1	11.192 km
2	0.01	1.1192 km
3	0.001	111.92 m
4	0.0001	11.192 m
5	0.00001	1.1192 m
6	0.000001	111.92 mm
7	0.0000001	11.192 mm
8	0.00000001	1.1192 mm

(a) HP ProLiant ML350p Gen8 [HP(2014b)]

(b) HP Pavilion Elite m9782pt [HP(2014a)]

Figure 3.12: New server that will be installed on the car and the old server that will be removed

Table 3.3: New and old server comparison

	HP ProLinant ML350p Gen8	HP Pavilion Elite m9782pt
Processor	Intel Xeon E5-2640 v2 (2.0Ghz / 8-core)	Intel i7 920 (2.66Ghz / 4-core)
Memory	16Gb	4Gb
Storage	1TB (10000rpm)	640Gb (7200rpm)

Chapter 4

Methods and Programming

This chapter describes how the system works and what was implemented. This concerns the several components and their interconnections and outputs.

4.1 Proposed Architecture

The main goal was to create an application that would calculate the route between two points and output the maneuvers that needed to be executed along it. The application should have a visual interface that would allow monitoring the car, choose a destination and show the maneuvers that needs to be executed. The idea was to split this problem into three modules, one that calculates the route, one to interface with the GPS and other modules and a third module that would be the user interface. In order to save time and have a precise solution for calculating a route, an open source software has been chosen. For the user interface a solution based on a webpage was chosen. The initial idea for the relationship between these modules is shown on figure 4.1, where WEB is the user interface; Navigator is the module responsible for communicating with the WEB and the Planner module; Planner is the module responsible for calculating the route. An example of what the application could be is represented on figure 4.2, where several waypoints are placed on the road and each of them has a specific maneuver.

The main framework used to develop the modules is ROS, and with it a client-server architecture that enables communication between two nodes only when needed, in this case the Navigator and the Planner modules. The main programming language used was C++. The OSRM-backend calculates the requested route, this program was installed on the LAR (Lab-

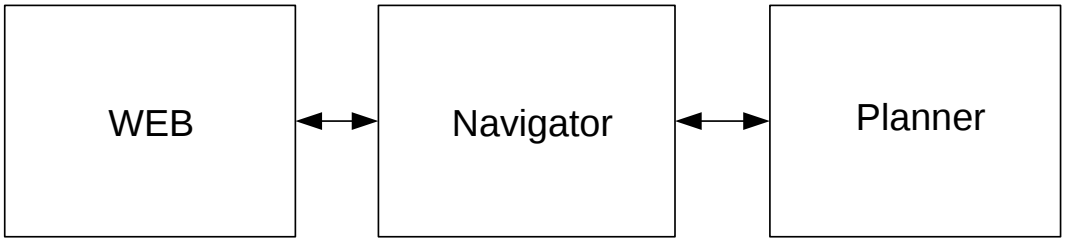


Figure 4.1: An initial solution to solve the problem was to split it into three modules

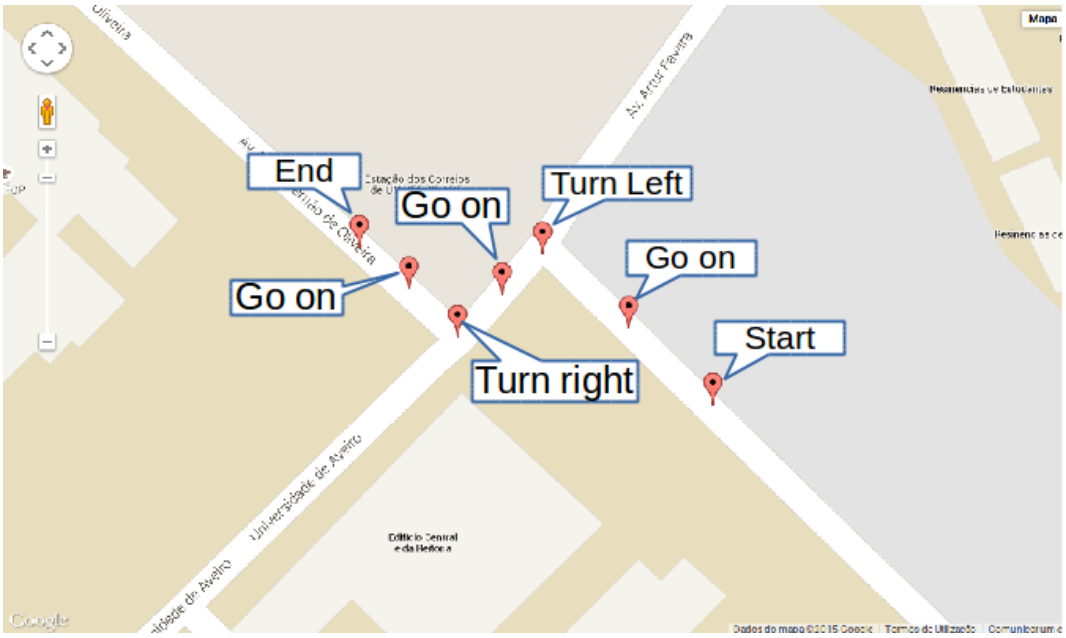


Figure 4.2: Application Example of possible solution, in which each waypoint, from the start till the end has an instruction associated

---

oratório de Automação e Robótica) server and can be used by anyone. The planner module will be responsible to communicate with this backend, it works based on web requests. Curl (Client URL Request Library) is also used to do a web request and gather the response, in this case in form of a JSON (JavaScript Object Notation) file. To store all the information for each mission, the database that is used is PostgreSQL 9.3. The web server used to host all the web modules is Apache2. PHP5 is used to communicate with the database and access all the relevant information. A package from GAVLab, called *novatel*, is used to establish a communication with the GPS receiver. The problem was split into three main parts:

Mission planner - Server based service
: responsible for communicating with the OSRM-backend; processes the response and returns a route based on the given start and destination coordinates.

Navigation manager - Client
: will process the user requests, write the waypoints, instructions, current position, closest waypoint and user destination to a Database, getting the coordinates from the GPS receiver;

Web page - Visual Interface
: reads the data that was written by the client from the database and presents the waypoints, the current and next instructions and the current position on the web page. The user can select the desired destination by clicking on the map. It also allows to track the vehicles movement.

The proposed architecture and the interaction between all modules and components is shown on figure 4.3

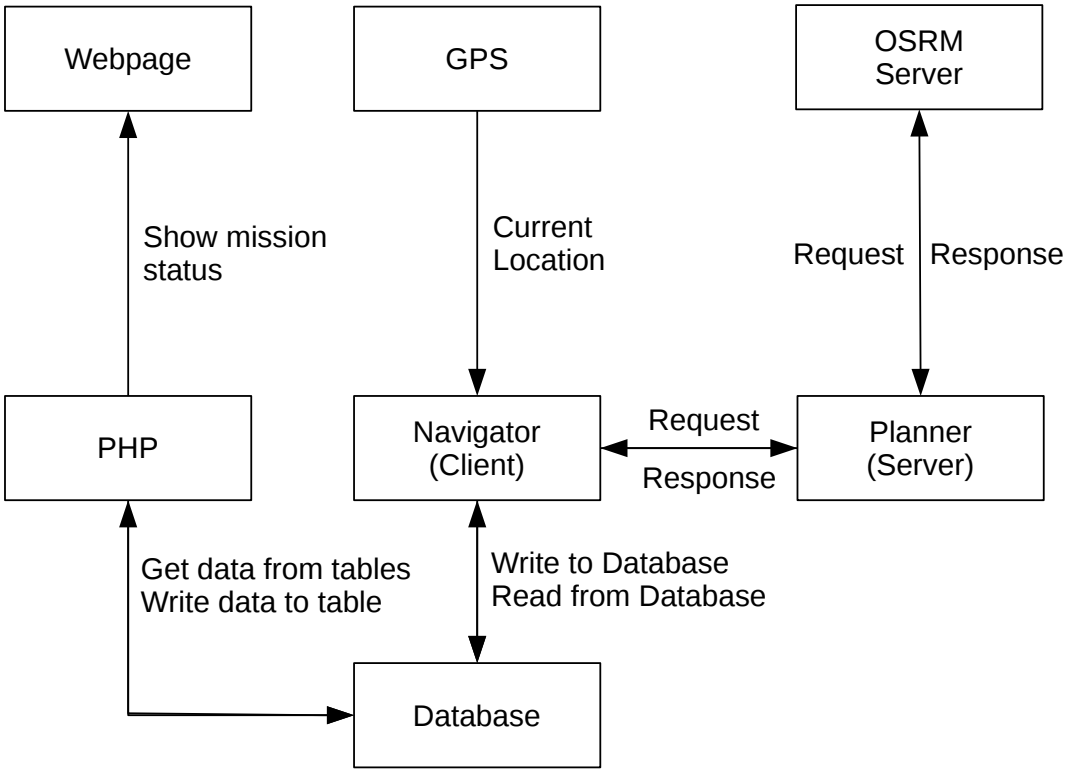


Figure 4.3: Interaction between all modules and components

4.1.1 Mission planner

This module will be the server and will process the requests from the client and returns a response. The current position and the desired destination coordinates are sent, and what is returned are the waypoints and all instructions. A custom ROS service has been implemented to send and receive this data. The client chooses a destination and this information is then sent to the server. When a request arrives, the server builds a specific URL (Uniform Resource Locator) that using curl will be sent to the OSRM-backend which, in case of a successful request, responds with a JSON containing all the data for the trip, an example of a response is shown in figure 4.4.

This data needs then to be processed in order to retrieve only the desired information, namely waypoints and instructions. This was done using the *JSONCpp library*, obtained from [Lepilleur(2015)]. But the waypoints come in and encoded format (Google Encoded Polyline), an example can be found in table 4.1, and therefore an algorithm to decode this information had to be written (Appendix C).

Luckily decoding this format is quite common and the Google company provides the instructions [Google(2015)], unfortunately no algorithm in C++ was found, so an algorithm written in JavaScript [Mapquest Inc.(2015)] had to be adapted to C++. After successfully decoding the waypoints, they had to be matched with the respective instruction. After this process, this module returns the waypoints and the instructions that need to be done during the entire trip. Figure 4.5 shows an example of the waypoints and the instructions that are returned by the this module.

Figure 4.4: Osrm-backend Response to a route request

Table 4.1: Polyline Example

Encoded Polyline	Decoded Polyline
`ez~vFvwys@o@~@NRTVU`	40.63155, -8.65676
	40.63179, -8.65708
	40.63171, -8.65718
	40.63160, -8.65730
	40.63171, -8.65745

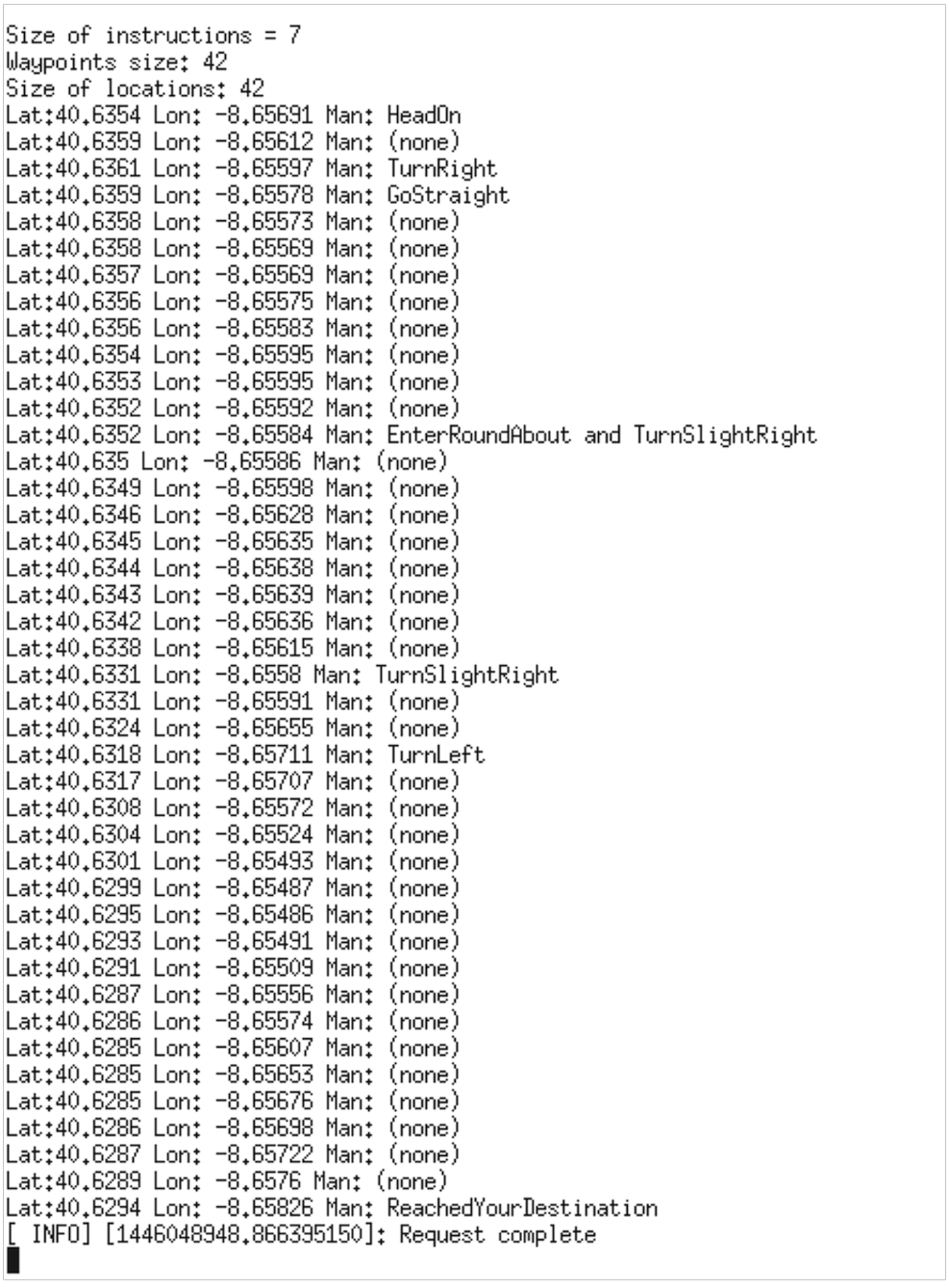


Figure 4.5: Mission Planner response showing a list of waypoints and the respective instruction

The following eighteen maneuvers are available:

- NoTurn
- GoStraight
- TurnSlightRight
- TurnRight
- TurnSharpRight
- UTurn
- TurnSharpLeft
- TurnLeft
- TurnSlightLeft
- ReachViaLocation
- HeadOn
- EnterRoundAbout
- LeaveRoundAbout
- StayOnRoundAbout
- StartAtEndOfStreet
- ReachedYourDestination
- EnterAgainstAllowedDirection
- LeaveAgainstAllowedDirection

An example of the mission planning workflow is illustrated in figure 4.6.

4.1.2 Navigation manager

This module will act as the client. It establishes a communication with the GPS receiver and communicates with a database. It subscribes to the messages that are being published by the GPS topic. When the navigation manager module starts, it establishes a communication with the database and, in case of a successful connection, it proceeds else the program stops and lets the user know that there was an error. After successfully establishing a connection it will delete all the data in all tables. A small menu in the console appears and waits for the user to select the desired destination from a predefined list of destinations or, by using the webpage, to choose a custom destination. The current position and the destination coordinates are then sent to the server as a request. After successfully obtaining the response, the destination coordinates, waypoints and instructions are written to the database. After this step the waypoints received need to be matched with the coordinates from the GPS. The GPS coordinates are also written to the database. Using the Haversine formula (4.1) [Wikipedia(2015)], the distance between two pairs of latitude and longitude coordinates can be calculated.

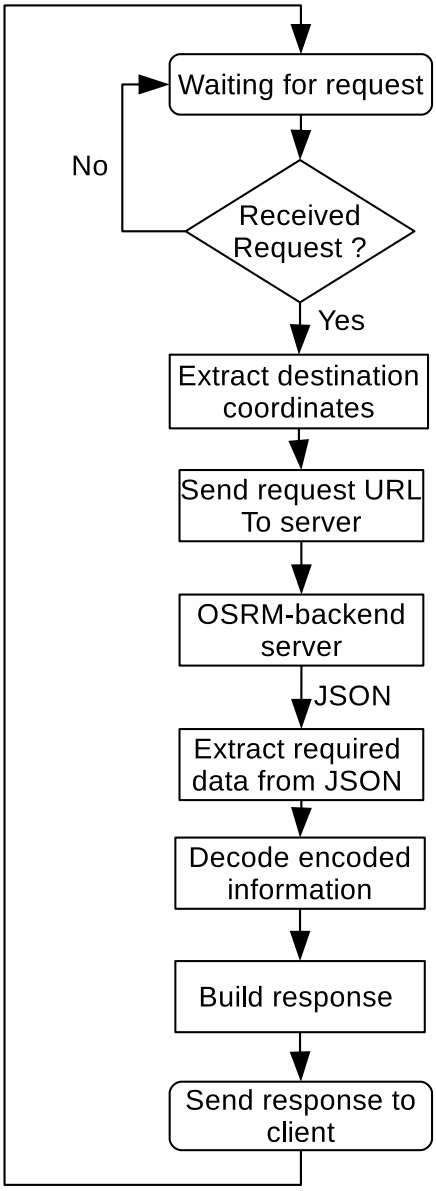


Figure 4.6: Mission planner workflow, illustrating all the steps since the request until a response is sent

d : is the distance between the two points (along a great circle of the sphere),

r : is the radius of the sphere (the mean radius of the Earth is used, which is 6371 km),

ϕ_2, ϕ_1 : latitude of point 1 and latitude of point 2 (in radians),

λ_2, λ_1 : longitude of point 1 and longitude of point 2 (in radians).

$$d = 2r \arcsin\left(\sqrt{\sin^2\left(\frac{\phi_2 - \phi_1}{2}\right) + \cos(\phi_1)\cos(\phi_2)\sin^2\left(\frac{\lambda_2 - \lambda_1}{2}\right)}\right) \tag{4.1}$$

The Haversine formula is used to calculate the distance between all waypoints from the list and the current GPS coordinates. The waypoint with the shortest distance is then chosen, and the instruction associated with it is presented. The procedure is similar for the next instructions to perform, but instead of choosing the closest waypoint it chooses the second closest waypoint. These two items are then written to the database. This module also monitors the movement of the car and guarantees that, in case it goes out of the planned route, a new route based from the current position to its previous destination is generated. This is done by comparing the two closest waypoints and, in case they both increase, an event is triggered and a new request is made to the server.

An example of the workflow is illustrated in the figure 4.7.

Database

The database system used is PostgreSQL 9.3. After installing PostgresSQL 9.3 it was necessary to create a user and a password. After that, it is necessary to create a database; it was named "*atlas_navigation*". Finally all five tables are created. The database contains five tables that will be used to store the data from the Navigation manager module and from the webpage. In the first table the waypoints and the instructions are stored. On the second table the current position acquired from the GPS receiver is stored. Table three contains the closest waypoint. The forth table contains the destination coordinates, and the last table contains the next instruction. The structure of the database is represented in figure 4.8.

The connection to the database is made using a localhost connection, as the database is stored in the computer aboard the Atlascar. A graphical interface, pgAdmin III, is also available to monitor the databases (figure 4.9).

4.1.3 Web page

The Webpage, figure 4.10, allows the user to choose the desired destination by clicking on the map and pressing the "submit" button. It presents to the user a visual interface of the route (the waypoints are drawn on google maps), the location of the car and the current and next maneuver that needs to be executed. It runs on a local Apache2 server. The server contains the webpage layout (.html)(Hypertext Markup Language), two javascript files (.js) and a PHP (Hypertext Preprocessor) file (.php).

The workflow of the webpage is described in figure 4.11.

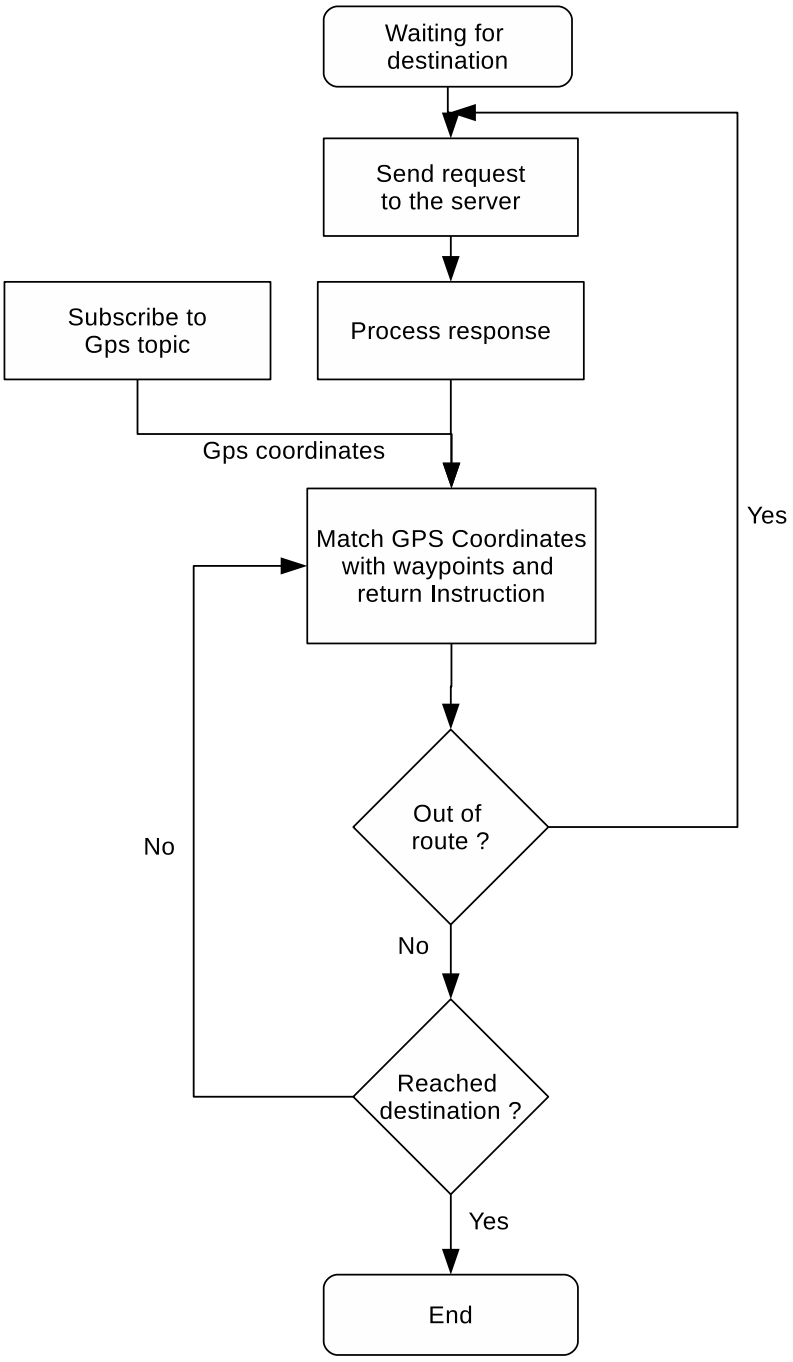


Figure 4.7: Navigation manager workflow, illustrating all the steps and decisions that are taken by this module

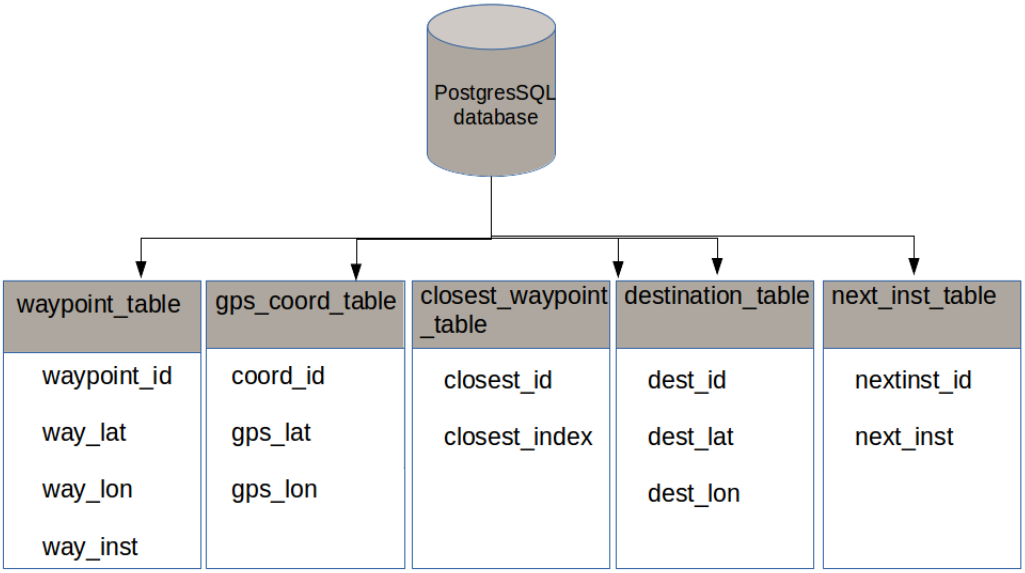


Figure 4.8: Database Structure. There are five tables with the mission information

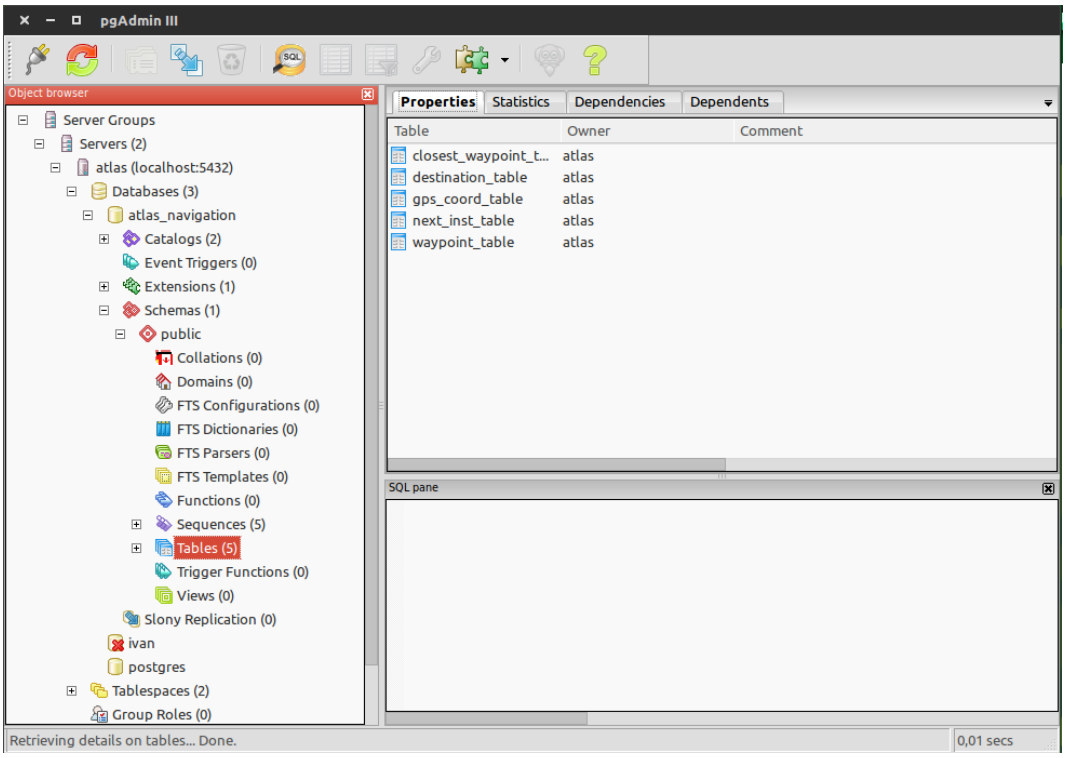


Figure 4.9: pgAdmin III interface that allows monitoring several databases and its content

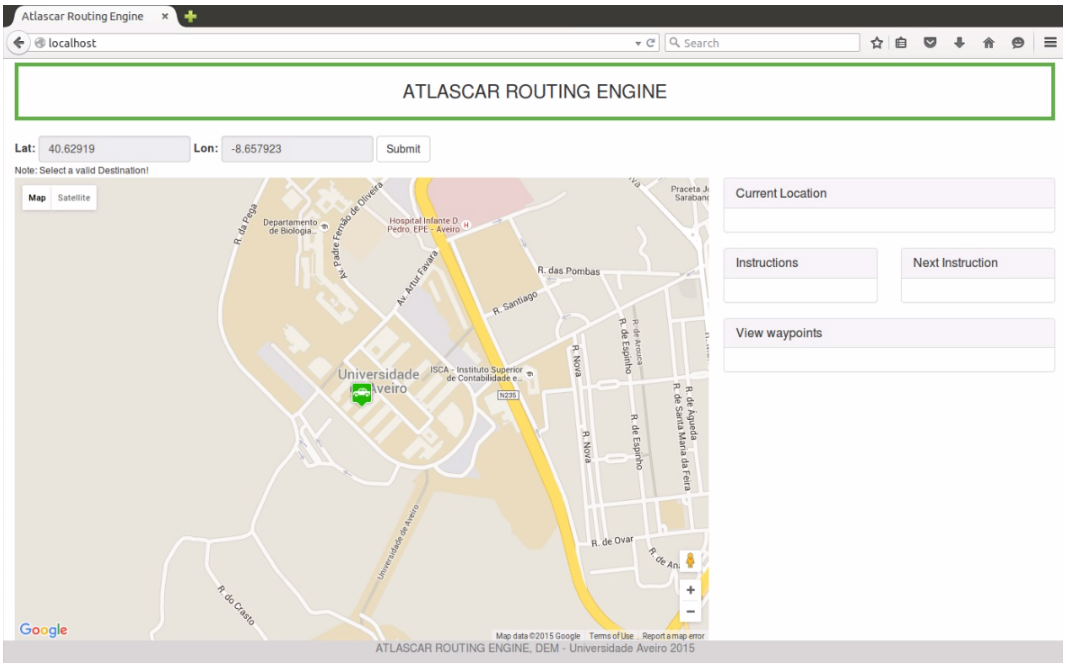


Figure 4.10: Webpage that provides a user interface and allows choosing a destination

---

Html

This file contains the layout of the webpage which was designed using the Bootstrap framework. The user is able to choose a custom destination by clicking on the map. The coordinates are then saved and when the user presses the "*submit*" button they are sent to the .php and stored in the database. The destination can also be chosen from the console after executing the "*mission_planning*" launch file. Using the google Maps API (Application Program Interface) the location of the car using a custom icon and also all the waypoints are drawn on the map. The current position, the maneuver that needs to be executed and the next maneuver to be executed is shown on the page. Every time the car approaches to the closest waypoint, this will change its default icon to a custom one.

PHP

It was necessary to install *php5* and the *php5-pgsql* package, which provides a module to establish connections to a PostgreSQL database directly from PHP scripts. By using PHP we establish then a connection to the "*atlas_navigation*" database (listing 4.1).

Listing 4.1: Database connection

```
//Connect to database
  $dbconn = pg_connect("host=localhost port=5432
  dbname=atlas_navigation user=atlas
  password=atlascar")or die(
  'Could not connect: ' . pg_last_error());
```

By using specific queries the data is retrieved from the tables. Listing 4.2 shows an example query that retrieves everything inside the "*waypoint_table*".

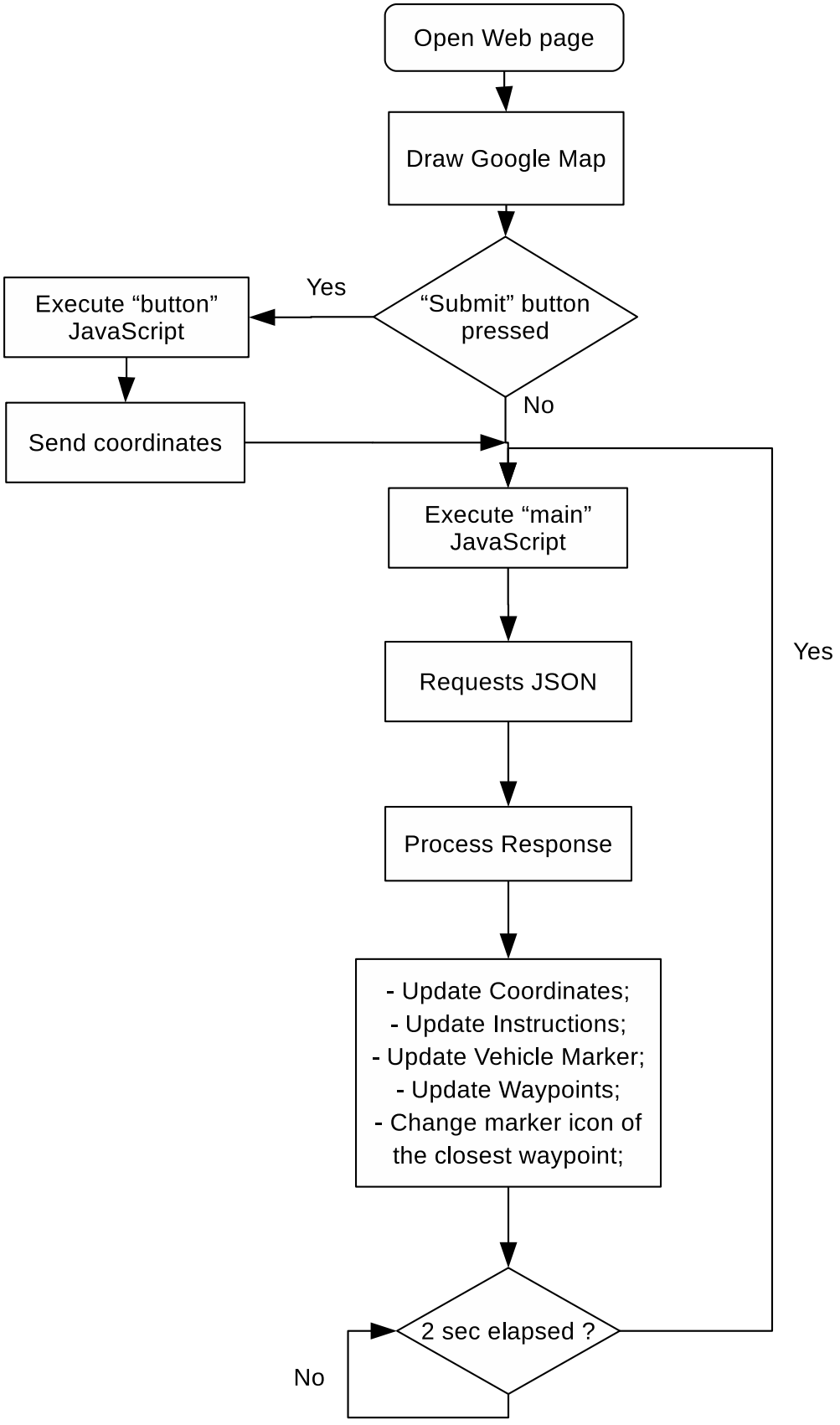


Figure 4.11: Webpage Workflow illustrating the steps and decisions that are taken by the user interface

Listing 4.2: Database query

```
//Performing SQL query
$query = 'SELECT * FROM waypoint_table';
```

After reading the data from the databse, the waypoints, instructions, closest waypoint and current position are processed and encoded in a JSON format. This JSON is then accessible to the Javascript. This module is also responsible for writing the desired destination coordinates to the database.

JavaScript

A Javascript program named "*main.js*" requests the JSON object that has been built by the PHP. After obtaining the JSON, the script accesses the information inside it, namely the waypoints, instructions, GPS coordinates and the closest waypoint. This is done using functions from the jQuery library. The data obtained from the JSON is then processed and shown on the webpage. Every two seconds the PHP is called to check for new data and update the results. A second Javascript named "*button.js*" is responsible for sending the desired destination coordinates to the PHP. This happens when the user presses the "*submit*" button. After successfully sending the destination a message appears informing the user that the destination has been sent, this information fades out after five seconds.

Chapter 5

Experiments and results

Several tests have been made to the software and to the hardware. First the software was tested in the laboratory, by doing simulations. After this, the software was tested in different paths, from city to open field. A video for every path was recorded for further analysis and all maneuvers were tested during these procedures. Invalid destinations where also tested and in case an unreachable location is chosen, the software returns the path to closest reachable location. The GPS receiver was also tested to see where it loses signal. This happens, for example, inside buildings (e.g. garage or underground parks) but only if the receiver is still trying to acquire coordinates. Because after this it even works in the garage. The system overall works well. But it might happen that it misses the current instruction to do, due to the fact that there was multiple waypoints close to each other. Even if it fails the current instruction to do, the next instruction to do was already shown so the user is aware of what he needs to do.

5.1 Fine Tune

The tests were made with different values to check if the car went out of route and to determine the next instruction that needs to be executed. First the system was tested with a value of 100m, this would allow the vehicle to move away 100m from the next waypoint it was supposed to go. If the vehicle moves away from the next waypoint above 100m the route would be recalculated. But this didn't work as well as expected so the distance was reduced to 75m which works much better.

The refresh rate for the webpage was also reduced from three seconds to two seconds, this allows a more accurate observation of the results, yet not perfect because the ideal scenario would be the same refresh rate as the publish rate of the GPS receiver. Since the GPS was set to publish coordinates each 0.25 seconds, some information is lost. This could be improved in the future, because there was not enough time to find, learn and adapt the new solution.

5.2 Missions

During the missions, figure 5.1 and 5.2, a rosbag has been created to record the coordinates for further analysis. This coordinates have then been plotted on the map.

On the first mission the original plan was to go from point A to B (red route) and test several maneuvers. But during this mission the original plan was ignored to test if the system

Figure 5.1: First mission to test the system. The route was from point A to B. The original route (red) was ignored at a certain point and a new route was generated (blue) from A1 to B1.

Figure 5.2: Second mission to test the system. Following the original route.

generates a new route within 75m. This new route (blue route) has been successfully generated (from point A1 to B1) as shown in figure 5.3. The distance was not 75m as expected, instead the distance obtained was 80m, which is still quite good.

The second mission was also randomly chosen but this time without ignoring the original plan.

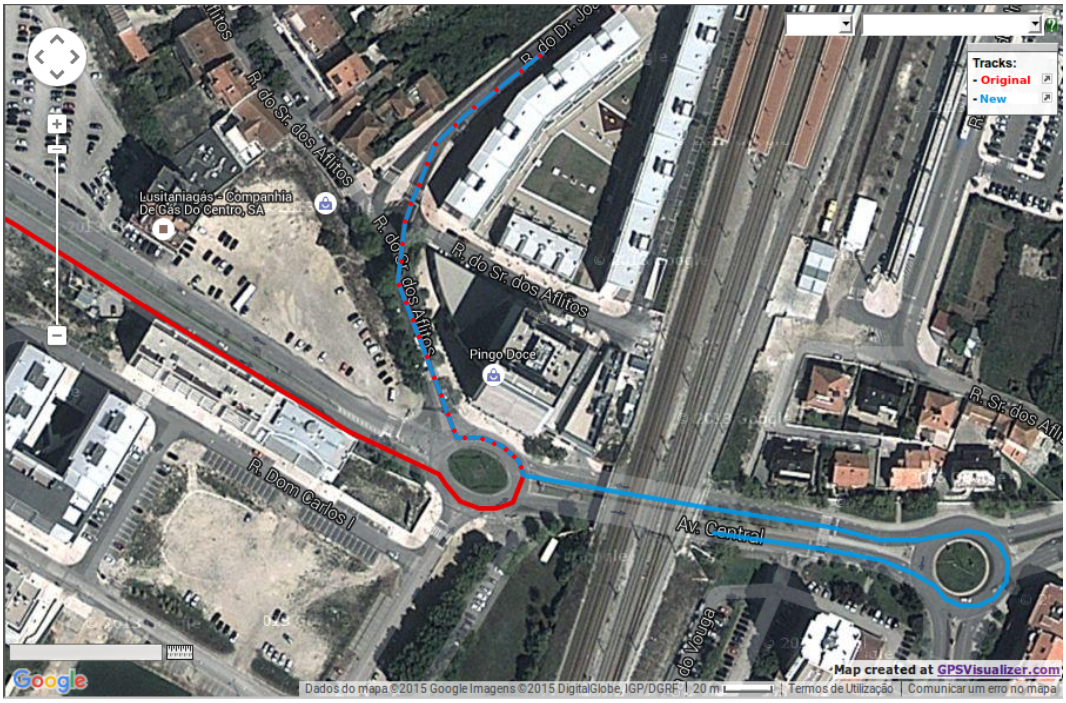


Figure 5.3: Alternative Route generated (blue) when the original route was ignored.

The GPS works very well in urban areas and even short tunnels are not a problem. A good example can be found in figure 5.4 which shows the recorded coordinates plotted on a map. Although there was a situation in both missions where the GPS coordinates where not so good, this happened in an area with a high density of trees. In this situation the GPS still works, but since its signal is being blocked, without loss if signal, the coordinates vary a bit and the result shown in figure 5.5 is not so good.

During these missions two different videos for both missions where recorded. One video contains the recordings from the webpage and the other one which was made with a camera contains the recordings from path made during the mission. These videos where used for further processing.

5.3 Instructions

On figure 5.6 the current instruction is to turn left and the next instruction is turn right, which corresponds to the reality as shown by the camera.

Another example of an instruction is shown in figure 5.7 where the user is informed to turn slight right.

When there is no instruction it means to go on/follow the road. There is also an instruction that might appear which tells the user to go straight and don't turn as shown in figure 5.8.

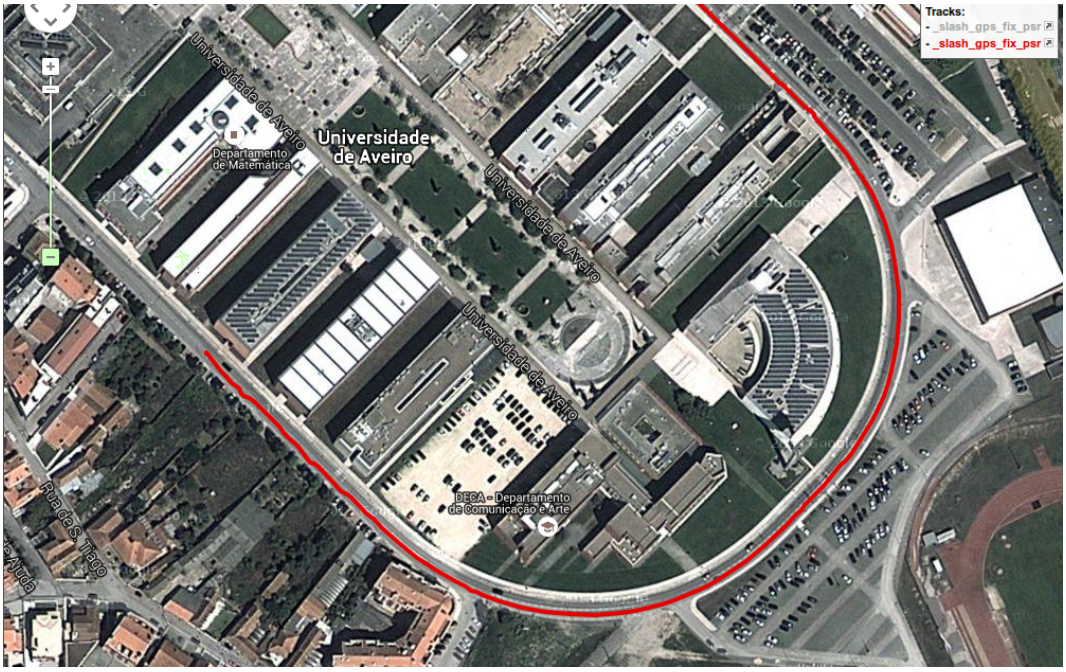


Figure 5.4: Good GPS Coordinates Recording

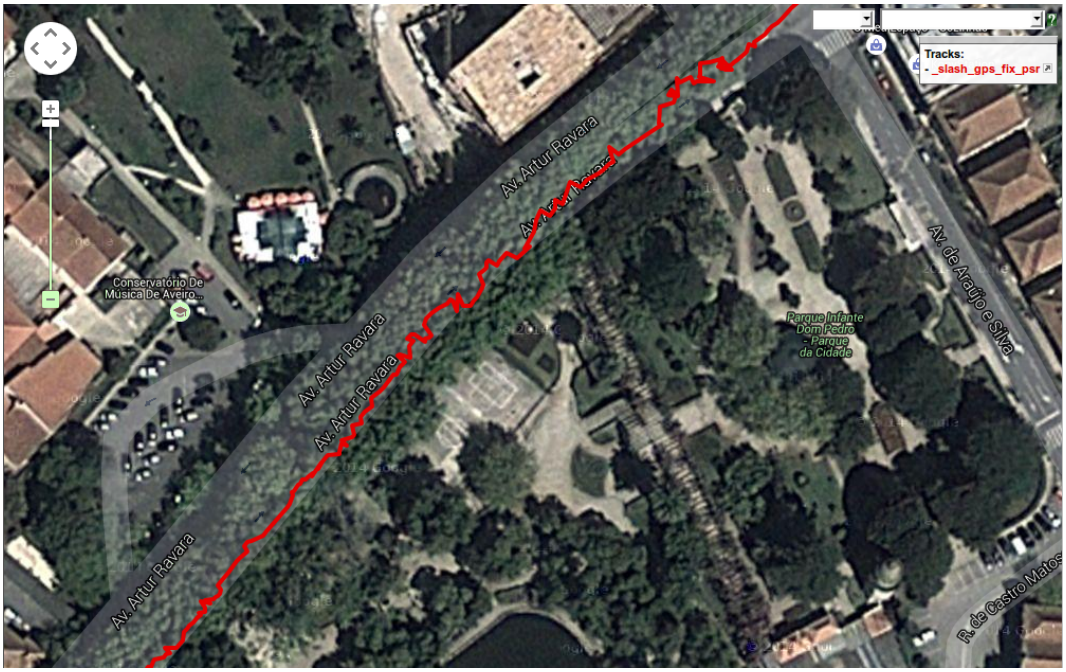


Figure 5.5: Bad GPS coordinates recording, due to the high density of trees.

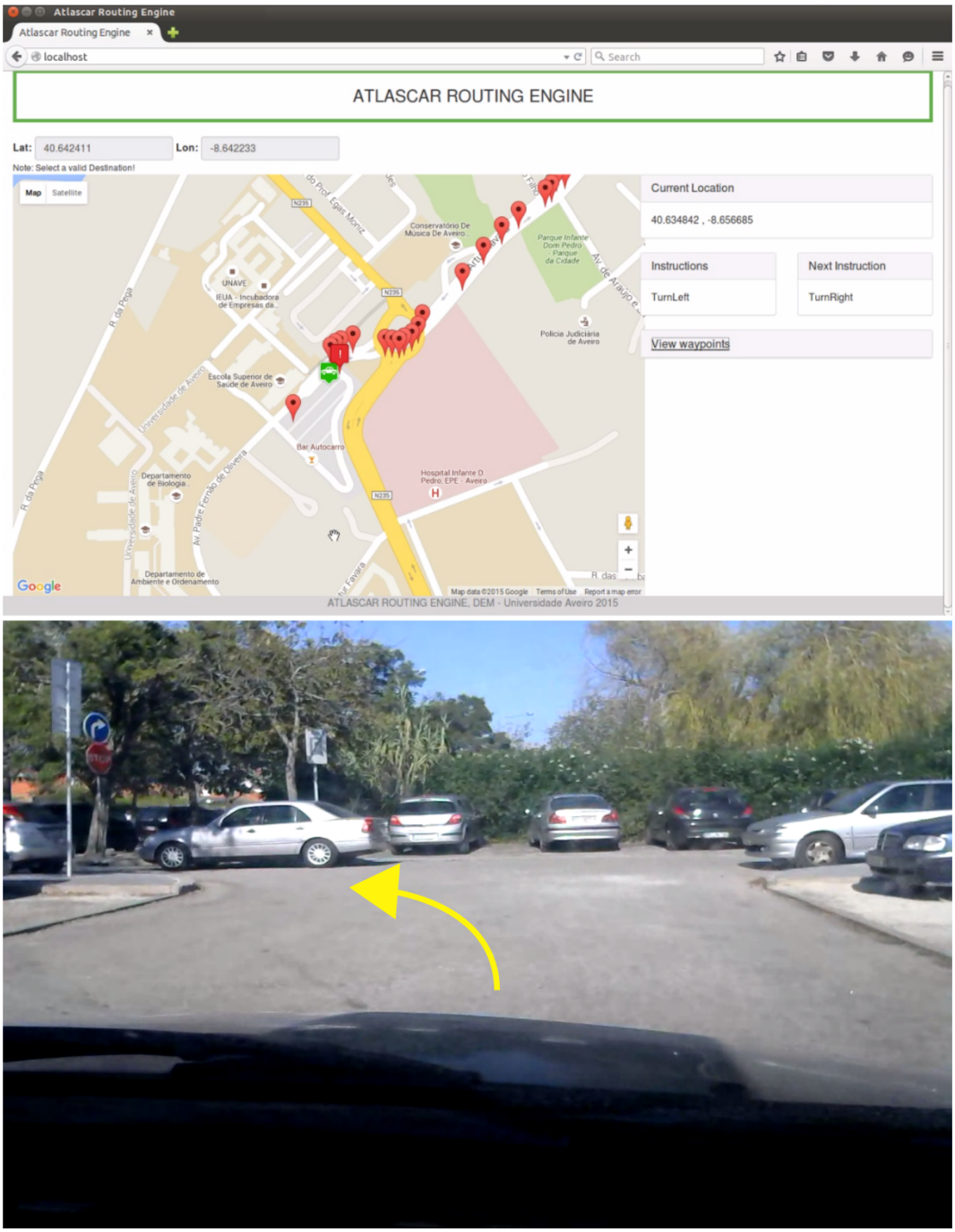


Figure 5.6: Instruction to turn first left and then right

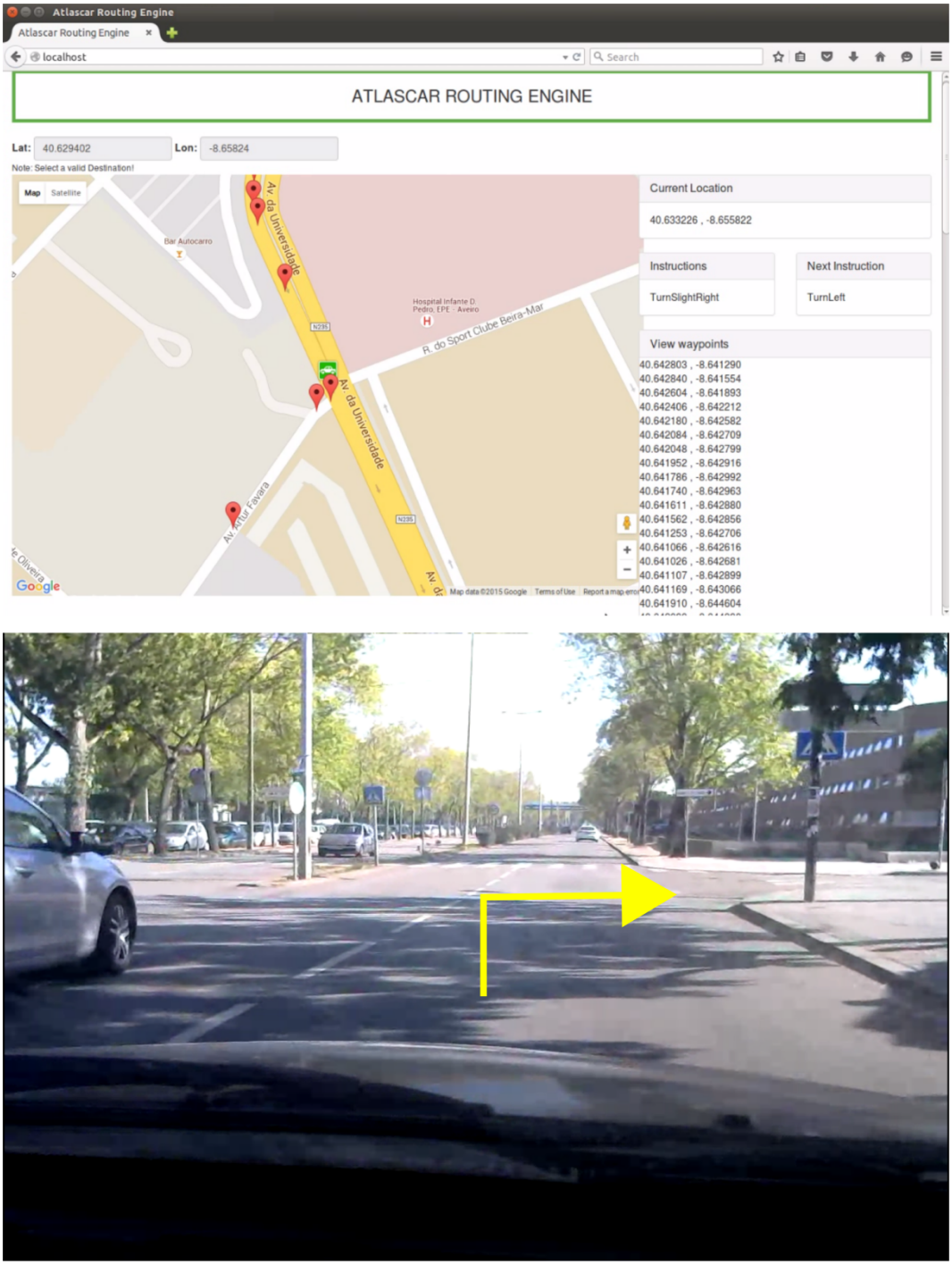


Figure 5.7: Slight right turn

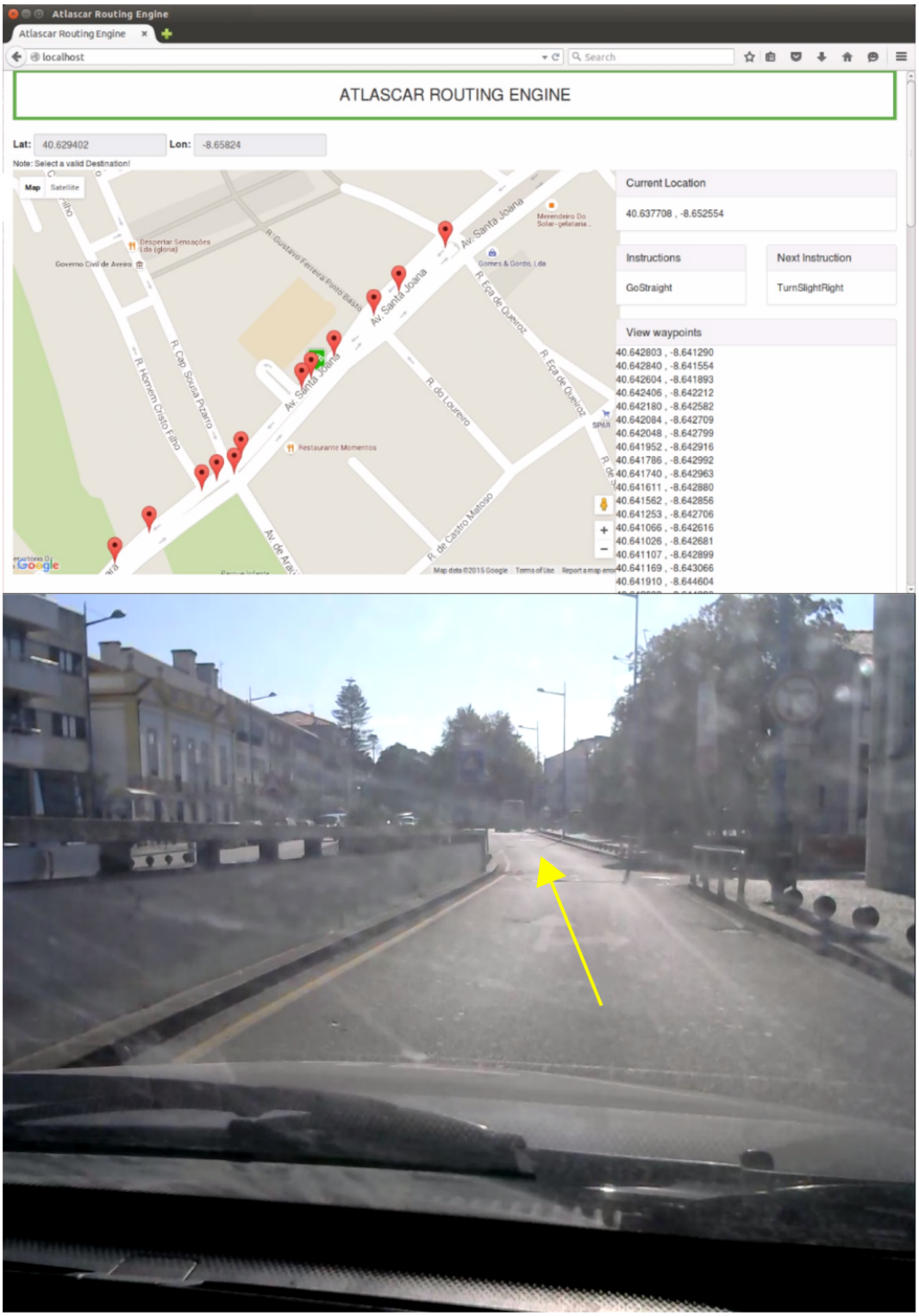


Figure 5.8: Instruction to follow road and ignore all exits

The instructions overall are quite good. But some instructions in roundabouts can be a bit more confusing for the user as shown in figure 5.9. In this case the goal was to go back, so the third exit after entering the roundabout needs to be taken. This is not so clear with the instruction given, which says "Enter Roundabout and turn right". But the instruction is not wrong and it appears on the correct exit.

A less confusing instruction for a roundabout is shown in figure 5.10. In this case the goal is to take the third exit, but the exit that leads to the hospital doesn't count, so the second exit needs to be taken. The instruction says "Enter Roundabout and turn slight right". If the goal is to take the first exit in a roundabout the instruction would be "TurnRight" so "Enter Roundabout and turn slight right" means to take the second exit and "Enter Roundabout and turn right" indicates to take the third exit.

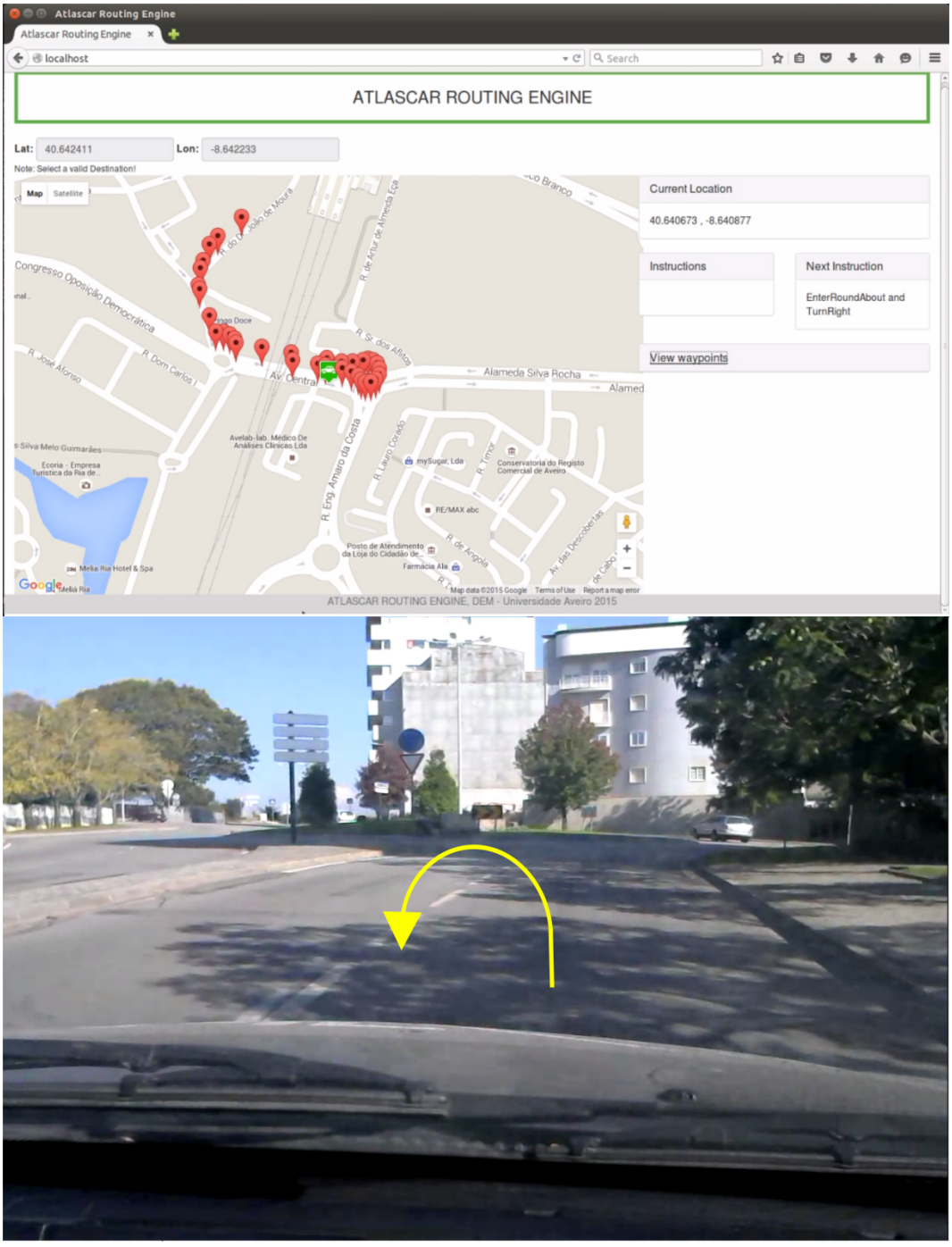


Figure 5.9: Confusing instruction to take the third exit at the roundabout

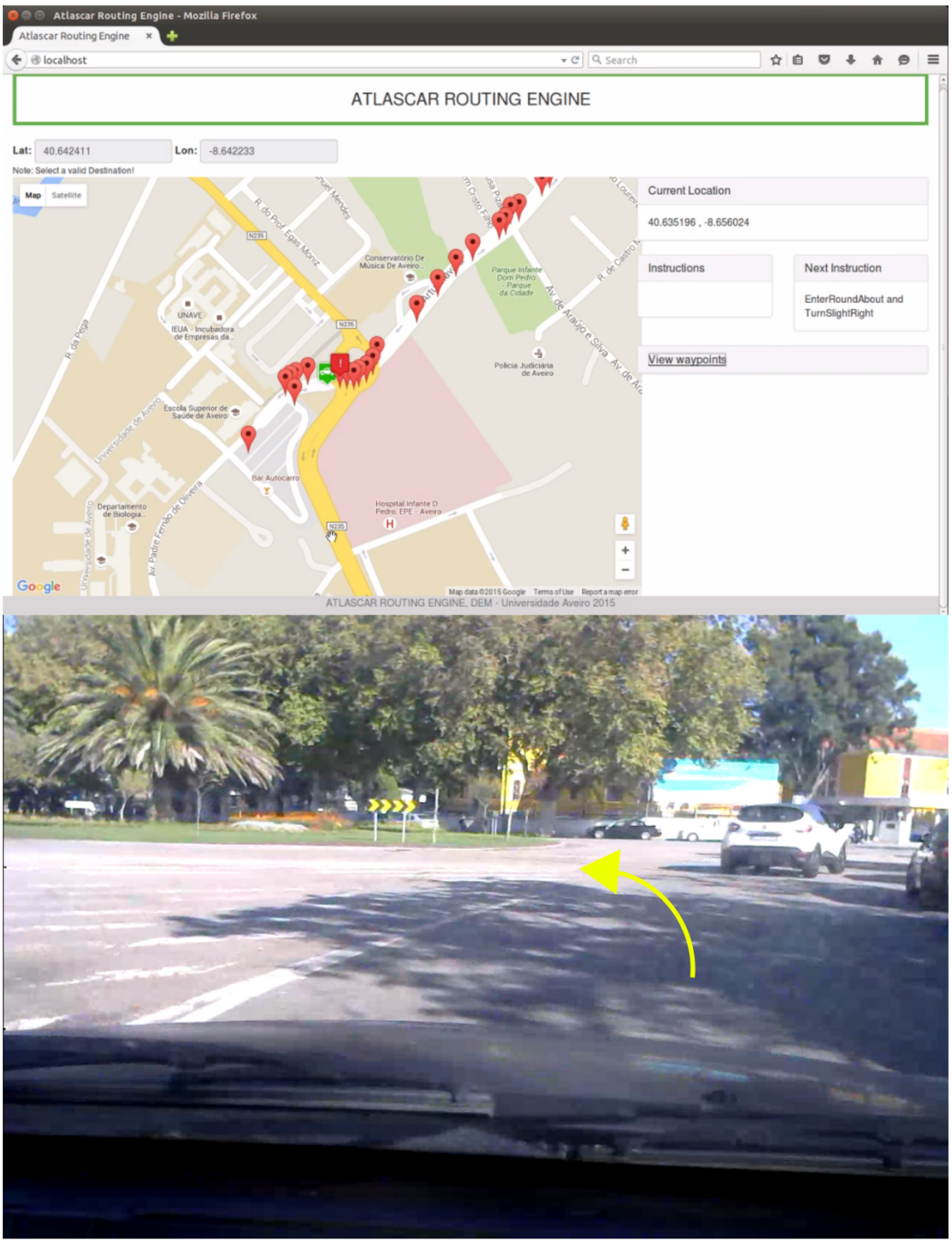


Figure 5.10: Roundabout instruction to take the second exit

Chapter 6

Conclusions and future work

6.1 Conclusion

The objectives for this thesis were to install the new hardware for the Atlascar, namely: a more powerful server and a high precision GPS; Developing a graphical interface that would allow to choose the start and destination coordinates in an interactive way, this application should also allow the user to monitor in real-time the steps of the mission; Developing a module to execute the mission planning that uses high level maneuvers or decisions that need to be executed during the mission. And finally, in an more advanced phase, this module should be able to select and execute the necessary maneuvers. This work contributed to the need the Atlascar had of a solution for mission planning and managing. The developments of this work contributed to fill the need for a mission planning and managing solution that would allow the car to become more autonomous.

The objectives proposed for this thesis were successfully achieved, tested and documented. An application to track the vehicle in real-time and output sequences of high level maneuvers (e.g., turn right, turn left, etc.) that need to be executed during the whole mission was successfully built. This was done using an open source routing planner to calculate the path between two locations, a high-precision GPS to determine the locations of the vehicle, a webpage to serve as a user interface and a database to store the data. All these solutions had to be linked to work in a ROS environment. The installation of the new hardware (server and GPS) was also done by redesigning the trunk layout and with some interventions on the equipment. Interfacing with the GPS was also achieved thanks to a ROS package created by the GPS Vehicle Dynamics Laboratory from the Auburn University (GAVLab).

The system works well and provides a user-friendly interface. Establishing a communication on linux with the GPS was complicated but, in the end, everything worked well. As expected, it is a very precise GPS and works very well in areas where a cheaper GPS could possibly fail. A solution to communicate, write and read from a database via C++ was also found and implemented, since the *database_interface* package available in the LAR repository doesn't work. It was not possible to install the package due to the errors it produced, also, including the header files did not work and caused a lot of errors. The backend provided by OSRM works very well and saved a lot of time that otherwise would be spent in writing algorithms for calculating the best way between two points. The instructions that are provided by this software are simple and easy to understand, except for the ones related to roundabouts where the instructions can be confusing. This needs to be fixed in future works.

This software was later installed on the server that is located in the laboratory. This service is now available for everyone to use, at the university or at home. In case there is a power failure or if the servers need to be rebooted, the backend needs to be launched again which is carefully explained in appendix A. The software infrastructure is working well and is ready for upgrades, improvements or modifications.

During the tests, there was an error that occurred several times and which precludes the correct operation of the system. With the help of both advisors the problem was found. It had to do with the power DC inverter that was causing interference in the USB ports. The problem was not deeply investigated but the new UPS (Uninterruptible Power Supply) that was bought apparently could solve this issue. The temporary solution found is to turn off the inverter and work only with the power from the UPS which lasts around 40 minutes.

6.2 Future work

Despite the fact that the system is working well, it still has a few things that could be improved. For future work:

- it is necessary to make this module communicate with the vehicle hardware and trigger the respective maneuvers;
- a new solution for the visual interface could be implemented to update the values without having to refresh the page every time;
- the server lacks a firewire card which needs to be installed in order to make the cameras work. The firewire card that needs to be installed is currently installed on the old server which is located in the laboratory;
- possibly finding a better way to install the software for the GPS since it was installed without creating a catkin package for it;
- the launcher file for the GPS could be improved so that it searches automatically for the USB port where the device is plugged in;
- install the new UPS to solve the interference problem;
- create a script for the LAR server that automatically launches the osrm-backend software.

References

[Almeida(2010)] Jorge Manuel Soares Almeida. *Target tracking using laser range finder with occlusion.* Master thesis, University of Aveiro, 2010. URL http://lars.mec.ua.pt/public/LARProjects/Perception/2010_JorgeAlmeida/Documents/Tese_MIEM_Jorge_Almeida.pdf.

[Andrews(2007)] Angus P Andrews. *Global Positioning Systems , Inertial Navigation And Integration.* John Wiley & Sons, Inc., second edi edition, 2007. ISBN 9780470041901. doi: 10.1002/0471200719.

[Azevedo(2014)] Rui Filipe Cabral Azevedo. *Sensor Fusion of LASER and Vision in Active Pedestrian Detection.* Master thesis, University of Aveiro, 2014. URL http://lars.mec.ua.pt/public/LARProjects/Perception/2014_RuiAzevedo/Dissertacao_final_50321.pdf.

[Baptiste(2015)] Jean Baptiste. Exclusive Interview: Ford CEO Expects Fully Autonomous Cars In 5 Years - Forbes. http://www.forbes.com/sites/jeanbaptiste/2015/02/05/exclusive-interview-ford-ceo-expects-fully-autonomous-cars-in-5-years/2/, 2015. URL http://www.forbes.com/sites/jeanbaptiste/2015/02/05/exclusive-interview-ford-ceo-expects-fully-autonomous-cars-in-5-years/2/.

[Bevly and Cobb(2010)] David M. Bevly and Stewart Cobb. *GNSS for Vehicle Control.* Norwood, MA, 2010. ISBN 159693302X. URL http://www.google.hr/books?hl=en&lr=&id=y2z8q-sfL9YC&pgis=1.

[Durrant-whyte(2001)] Hugh Durrant-whyte. A Critical Review of the State-of-the-Art in Autonomous Land Vehicle Systems and Technology. *Sandia National Laboratories*, 2001. URL http://prod.sandia.gov/techlib/access-control.cgi/2001/013685.pdf.

[Geofabrik(2015)] Geofabrik. OpenStreetMap Data Extracts, 2015. URL http://download.geofabrik.de/.

[Geographyworldonline(2008)] Geographyworldonline. Latitude-Longitude Map.jpg, 2008. URL http://geographyworldonline.com/tutorial/latitudelongitude.jpg.

[Google(2015)] Google. Encoded Polyline Algorithm Format — Google Maps APIs — Google Developers, 2015. URL https://developers.google.com/maps/documentation/utilities/polylinealgorithm.

[GraphHopper(2015)] GraphHopper. GraphHopper Directions API with Route Optimization, 2015. URL https://graphhopper.com/.

[Hewlett Packard(2015)] Hewlett Packard. HP ProLiant ML350p Generation 8 (Gen8) QuickSpecs, 2015. URL http://www8.hp.com/h20195/v2/GetDocument.aspx?docname=c04128239.

[HP(2014a)] HP. PC Desktop HP Pavilion Elite m9782pt, 2014a. URL http://support.hp.com/pt-pt/product/HP-Pavilion-Elite-m9000-Desktop-PC-series/3953004/model/4000500/product-info.

[HP(2014b)] HP. HP ProLiant ML350p Gen8 Server, 2014b. URL http://h22150.www2.hp.com/D46D5D2B-905A-42D0-8FCC-517CECEB4B3D/index.html#overview.

[IEEE(2012)] IEEE. IEEE - News Releases, 2012. URL http://www.ieee.org/about/news/2012/5september_2_2012.html.

[James(2013)] Barker James. An Overview of the State of the Art in Autonomous Vehicle Technology and Policy, 2013. URL http://www.law.washington.edu/Clinics/Technology/Reports/AutonomousVehicles.pdf.

[Jeffrey(2010)] Charles Jeffrey. *An Introduction to GNSS: GPS, GLONASS, Galileo and other Global Navigation Satellite Systems*. NovAtel Inc., Alberta,Canada, first edition, 2010. ISBN 9780981375403.

[Knight(2015)] Will Knight. Vehicle-to-Vehicle Communications Will Save Lives on the Road — MIT Technology Review, 2015. URL http://www.technologyreview.com/featuredstory/534981/car-to-car-communication/.

[Lepilleur(2015)] Baptiste Lepilleur. jsoncpp library, 2015. URL https://github.com/open-source-parsers/jsoncpp.

[Lopez(2015)] Roger Lopez. SwRI: Autonomous Vehicle Navigation, 2015. URL http://www.swri.org/4org/d14/aerospace/path/auto.htm.

[Mapbox(2015)] Mapbox. Mapbox, 2015. URL https://www.mapbox.com/.

[MapQuest(2015)] MapQuest. MapQuest Developer Network, 2015. URL https://developer.mapquest.com/.

[Mapquest Inc.(2015)] Mapquest Inc. MapQuest Platform Web Services : Compressed Lat/Lng Encoding/Decoding, 2015. URL http://www.mapquestapi.com/common/encodedecode.html#js.

[McBride(2008)] James McBride. State-of-the-Art Autonomous Vehicle Technology When I Joined Ford Motor Company... page 34, 2008. URL http://www.fieldrobotics.org/icra08workshop/Program_files/12_IVS.pdf.

[Morais(2014)] Ricardo Morais. *Parametrização de Algoritmos para Deteção de Estrada a Bordo do ATLASCAR*. Master thesis, University of Aveiro, 2014. URL http://lars.mec.ua.pt/public/LARProjects/Perception/2014_RicardoMorais/Disserta%C3%A7%C3%A3o/dissertation_v0.pdf.

[NIIMS(2007)] NIIMS. Basic Land Navigation. Technical report, National Interagency Incident Management System, 2007. URL http://www.nwcg.gov/pms/pubs/475/PMS475_chap3.pdf.

[Nokia(2015)] Nokia. Build applications with HERE Maps API and SDK Platform Access - HERE Developer, 2015. URL https://developer.here.com/.

[Novatel(2014a)] Novatel. Combined GNSS/Inertial Navigation Systems . jpg, 2014a. URL http://www.novatel.com/an-introduction-to-gnss/chapter-4-advanced-gnss-concepts/combined-gnss-inertial-navigation-systems/.

[Novatel(2014b)] Novatel. Novatel SPAN-IGM-A1, 2014b. URL http://www.novatel.com/assets/Web-Phase-2-2012/Product-Page-Images/Product-Images-Banner-and-Thumbnail/SPAN/_resampled/SetWidth360-IGM-A1.png.

[Novatel(2015)] Novatel. OEM4 & OEMV USB Driver for Linux, 2015. URL http://www.novatel.com/support/known-solutions/oem4-and-oemv-usb-driver-for-linux/.

[Oliveira(2012)] André João Lopes Oliveira. *Sistema de Monitorização da Condução de um Automóvel*. Master thesis, University of Aveiro, 2012. URL http://lars.mec.ua.pt/public/LARProjects/Perception/2012_AndreOliveira-@Artur/Tesefinal/Andre_thesys.pdf.

[OpenStreetMap(2015)] OpenStreetMap. Planet OSM, 2015. URL http://planet.openstreetmap.org/.

[Osborne(2013)] Peter Osborne. The Mercator Projections. page 254, 2013. URL http://www.mercator99.webspace.virginmedia.com/.

[Pereira(2012)] Joel Filipe Pereira. *Autonomous parking using 3D perception*. Master thesis, University of Aveiro, 2012. URL http://lars.mec.ua.pt/public/LARProjects/RobotNavigation/2012_JoelPereira/Tese_escrita/Autonomous_parking_using_3D_perception.pdf.

[Pinho(2014)] Sérgio António Matos Pinho. *Caixa Automática e Manobras Especiais no ATLASCAR*. Master thesis, University of Aveiro, 2014. URL http://lars.mec.ua.pt/public/LARProjects/HardwareInterfaces/2014_SergioPinho/documento/tese.pdf.

[Ramalhinho(2011)] João Ramalhinho. Accionamento Automático do Acelerador de um Veículo. Technical report, University of Aveiro, Aveiro, Portugal, 2011. URL http://lars.mec.ua.pt/public/LARProjects/HardwareInterfaces/2011_JoaoRamalhinho/relatorio_acelerador_ATLASCAR.pdf.

[Rocha(2011)] Tiago Nunes Rocha. *Piloto Automático para Controlo e Manobras de Navegação do AtlasCar*. Master thesis, University of Aveiro, 2011. URL http://lars.mec.ua.pt/public/LARProjects/HardwareInterfaces/2011_TiagoRocha/Tese/dissertacao_final_c_anexos.pdf.

[Santos and Almeida(2010)] V. Santos and J. Almeida. ATLASCAR - technologies for a computer assisted driving system on board a common automobile. In *13th International IEEE Conference on Intelligent Transportation Systems*, pages 1421–1427. IEEE, September 2010. ISBN 978-1-4244-7657-2. doi: 10.1109/ITSC.2010.5625031. URL http://www.researchgate.net/publication/224190645_ATLASCAR_-_Technologies_for_a_computer_assisted_driving_system_on_board_a_common_automobile.

[Silva(2013)] Pedro Batista e Silva. *Visual Pedestrian Detection using Integral Channels for ADAS*. Master thesis, University of Aveiro, 2013. URL http://lars.mec.ua.pt/public/LARProjects/Perception/2013_PedroSilva/Disserta%C3%A7%C3%A3o/tese.pdf.

[USGlobalSat Inc.(2014)] USGlobalSat Inc. USGlobalSat Inc. GPS-353-BU, 2014. URL http://usglobalsat.com/p-62-bu-353-w.aspx#images/product/large/62.jpg.

[Wikipedia(2015)] Wikipedia. Haversine formula, 2015. URL https://en.wikipedia.org/wiki/Haversine_formula.

Appendix A

General Instructions

A.1 Qt Installation

An example can also be found under the following link:
`http://sysads.co.uk/2014/05/install-qt-5-3-ubuntu-14-04/`

Go to the qt website, then choose downloads and select "Qt Online Installer for Linux 64-bit (29 MB)"

After downloading open the console and "cd" into the location of the downloaded file. Next do:
`chmod+xqt-unified-linux-x64-2.0.2-1-online.run` and then:
`./qt-unified-linux-x64-2.0.2-1-online.run`

Install the program.

To execute the program open the console and type:
`cd/home/<user>/Qt/Tools/QtCreator/bin/`

For example:
`cd/home/atlas/Qt/Tools/QtCreator/bin/`

After that type:
./qtcreator

In case you don't do the opening this way, it will cause errors when compiling programs that uses ROS.

A.2 Postgresql 9.3 Installation

An example can also be found under the following link:
`http://ubuntuhandbook.org/index.php/2014/02/install-postgresql-ubuntu-14-04/`

To get install PostgreSQL on Ubuntu 14.04, do the following steps:

1. Create and edit the PostgreSQL repository by running the command below:
   `sudovi/etc/apt/sources.list.d/pgdg.list`

   Press I on keyboard and add the below line into the file:
   `debhttp://apt.postgresql.org/pub/repos/apt/trusty-pgdgmain`

   Press Esc on keyboard and followed by :wq to save the file.

2. Download & import the repository key:
   `wget--quiet-O-https://www.postgresql.org/media/keys/ACCC4CF8.asc|sudoapt-keyadd-`

3. Update your system:
   `sudoapt-getupdate&&sudoapt-getupgrade`

4. Now you're able to install PostgreSQL via below command:
   `sudoapt-getinstallpostgresql-9.3pgadmin3`

A.2.1 Configuring the Database

An example can also be found under the following link:
`https://wixelhq.com/blog/how-to-install-postgresql-on-ubuntu-remote-access`

1. First, we'll switch to our postgres super user account: "sudo su - postgres"
   You are now in the postgres user account session, next we'll log into PostgreSQL itself using the psql CLI.

2. Now lets create a new role (New Database user)
   type: `createuser--interactive`

   Then answer the upcoming questions like this:
   Name of role: atlas
   new role will be a superuser?: n
   new role will be allowed to create databases?: y
   new role will be allowed to create more new roles?: y

3. Next, we will create a database with the name atlas_navigation
   createdb `<databasename>`

   `<databasename>` will be "atlas_navigation"

4. Last thing here would be to change ownership of the newly created database to your new role:

   Inside the console type : "psql"

   The command to change a owner of a database is: ALTER DATABASE `<databasename>` OWNER TO `<rolename>`;

   ALTER DATABASE atlas_navigation OWNER TO atlas;

   The user you just created has no password so you can't connect remotely using it. We'll need to set a password for this account before proceeding:

   The command to set a password for a database is : ALTER USER `<username>` WITH PASSWORD '`<newpassword>`';

   ALTER USER atlas WITH PASSWORD 'atlascar';

   Now enter \q and then "exit" to logout from postres user.

A.2.2 Connect to your Database Remotely

There are several ways to connect to your database server, here are the most common options:

1. Connecting to PostgreSQL using a connection string
   `postgresql://<username>:<password>@<server>:5432/<database>`
2. Connecting to PostgreSQL via the Command Line
   psql -h `<server>` -U `<username>` -d `<database>` -W

   The -W tells psql that you will be entering a password.

   In this case we will use option 2:

   psql -h localhost -U atlas -d atlas_navigation -W

Now that you are in the postgresql command line, here are some useful commands you can use:

\password Change your password

\q Exit psql

\l List all databases accessible by the current account

\du List roles

\c `<dbname>` Connect to a database

\dt List all tables in a connected database

\d `<table>` List columns in the selected table

\conninfo Show information about your current connection

Next we want to create the following tables, type:
CREATE TABLE waypoint_table (waypoint_id serial PRIMARY KEY, way_lat numeric, way_lon numeric, instruction varchar);
CREATE TABLE closest_waypoint_table (closest_id serial PRIMARY KEY, closest_index numeric);
CREATE TABLE gps_coord_table (gps_coord_id serial PRIMARY KEY, gps_lat numeric, gps_lon numeric);
CREATE TABLE destination_table (dest_id serial PRIMARY KEY, dest_lat numeric, dest_lon numeric);
CREATE TABLE next_inst_table (nextinst_id serial PRIMARY KEY, next_inst varchar);

POSTGRES should now be set up. Type \q to exit psql command line

You can use now pgAdmin3 as a GUI to watch your database. Just press"connect" fill in the fields like this:

Name: atlas

Host: localhost

Port: 5432

Service:

Maintenance DB: postgres

Username: atlas

Password: atlascar

And now press OK. Now you are connected to the atlas database, you can press the "+" to expand `Atlas->Databases->atlas_navigation->Schemas->public->Tables` .

To disconnect right-click "atlas(localhost:5432)" and choose "Disconnect/Connect Server"

A.3 Installing PHP5 and Apache on Ubuntu

Further examples can be found under the following links:
`http://www.howtogeek.com/howto/ubuntu/installing-php5-and-apache-on-ubuntu/`
`http://php.net/manual/en/pgsql.installation.php`

From a command shell, you will run the following commands:
sudo apt-get install apache2
sudo apt-get install php5
sudo apt-get install libapache2-mod-php5
sudo apt-get install php5-pgsql
sudo /etc/init.d/apache2 restart

After successfully installing these packages create an empty file named "index.php" edit and paste this inside

`<?php print_r(phpinfo()); ?>`

Copy this file to `/var/www/html` you will need sudo permissions so do: "sudo cp index.php /var/www/html"

Test apache2 by opening a new page in your browser and type "localhost" Test php by opening a new page in your browser and type "localhost/index.php"

To start and stop apache2 server type in console: sudo service apache2 start or stop

A.4 GPS Configuration

As described by novatel under the following link: `http://www.novatel.com/support/known-solutions/oem4-and-oemv-usb-driver-for-linux/`

In order to make the gps usb port be recognized by your computer it is necessary to create a new rule in the system.

sudo nano /etc/udev/rules.d/z90_novatel.rules
place this inside:

SUBSYSTEM==”usb”, SYSFSidProduct==”0100”, SYSFSidVendor==”09d7”, PROGRAM=”/sbin/modprobe usbserial vendor=0x09d7 product=0x0100”

BUS==”usb”, SYSFSidProduct==”0100”, SYSFSidVendor==”09d7”, SYSFSproduct==”NovAtel GPS Receiver”, SYSFSmanufacturer==”NovAtel Inc.”, SYMLINK+=”gps%n”

Save file. S

You will also need to add the computer `<user>` to the dialout group.
Open terminal and type :
”cd /etc/”
”more group”
Now check if dialout has no user added , or if the current user is added. If no user is added it should be like this: ”dialout:x:20:”

Now in the terminal type: ”sudo gedit group”

Edit the file so that you add your user to the dialout group:
”dialout:x:20:`<user>`”

Save file.

Log out from current session and enter again.
Voilá it should work now. Communication can be tested with CuteCom .

A.5 How to install osrm-backend

Tutorials available at:

`http://dogeo.fr/osrm-installation/`
`https://www.digitalocean.com/community/tutorials/how-to-set-up-an-osrm-server-on-ubuntu-14-04`
`https://github.com/Project-OSRM/osrm-backend/wiki/Building-on-Ubuntu` (1st) (the dependencies where not enough aparently ... , use the ones from digitalocean)
`https://github.com/Project-OSRM/osrm-backend/wiki/Building%20OSRM` (2nd)
`https://github.com/Project-OSRM/osrm-backend/wiki/Running-OSRM`(3rd)

This tutorial will guide you to the process of installing osrm-backend on your machine:

1. Enter on your Ubuntu account (make sure you have sudo permissions because you will need them)
2. Install dependencies

sudo apt-get install build-essential git cmake pkg-config libbz2-dev libstxxl-dev libstxxl-doc libstxxl1 libxml2-dev libzip-dev libboost-all-dev lua5.1 liblua5.1-0-dev libluabind-dev libluajit-5.1-dev libtbb-dev

if you get an error on step 3 at cmake try these additional dependencies:

sudo apt-get install libboost-all-dev libtbb-dev liblua5.2-dev libluabind-dev libstxxl-dev libxml2 libxml2-dev libosmpbf-dev libbz2-dev libprotobuf-dev

3. now do these steps:

   git clone `https://github.com/Project-OSRM/osrm-backend.git`
   cd osrm-backend
   mkdir -p build
   cd build
   cmake .. (if you get an error here its because of the dependencies, read step 2)
   make

4. Now download the maps that you want, in this case we will use the ones from Portugal. Go to `http://download.geofabrik.de/europe.html` and select Portugal from the list and download the .osm.pbf file. Copy this file into the "build" folder.

5. Now inside the build folder do these two steps:

   ln -s osrm-backend/profiles/car.lua profile.lua
   ln -s osrm-backend/profiles/lib

6. now do ./osrm-extract portugal-latest.osrm.pbf

7. wait...

8. now do ./osrm-prepare portugal-latest.osrm

9. wait ...

10. After completing these two steps lets change the ip. Do ./osrm-routed –ip=computerIp

11. Now lets test the installation and start the service, do ./osrm-routed portgual-latest.osrm

12. It should be working fine if you don't see any errors. You can test the system for example with this query :

    `http://193.137.172.18:5000/viaroute?loc=40.631599,-8.657289&loc=40.637814,-8.658555&instructions=true`

    you should get a similar message from figure 4.4

    You can see on the console, where the service is running, from where the request came from (ip and browser) and what was requested.

    Note: To shutdown the system simply do CTRL+C and to start it again do step 11. This way it will only work during this session if you logout this won't work anymore. In order to make this work even after you logout, you need to add "& disown".

    To do so, first make sure you are in the "build" folder (cd osrm-backend/build) then do:
    ./osrm-routed portgual-latest.osrm & disown

13. Enjoy !

A.6 How to use the program

After having everything launched , on the client console you can select one of the predefined options (1 to 8) or select a custom destination by clicking on the map. If you wish to select a custom destination don't press 8 now! First open the browser and type localhost. Then click on the destination you want to go, you will see a red marker on that location. After this, press "send location" and confirm the message that will pop-up. Now you can go back to the console and press 8. You are ready to start the mission. To select a different mission, do CTRL+C on the client console and launch it again. Repeat the steps.

How to execute the software step-by-step
There are two ways to do this, either by launching the components one by one.

1. sudo service apache2 start
2. open webpage (type in browser : localhost, wait until the map loads before proceeding to the next step. Refresh page if necessary)
3. roscore
4. roslaunch novatel novatel_for_psr.launch
5. rosrun mission_planning mission_planning_node
6. rosrun mission_planning mission_planning_client
7. select an option from the console and you are ready to go, results appear on the webpage

Or by launching the launch file, this will save you some time.

1. sudo service apache2 start
2. open webpage (type in browser : localhost)
3. roslaunch mission_planning mission_planning.launch
4. select an option from the console and you are ready to go, results will appear on the webpage. If you want to select a destination from the map, click on the desired location (a marker will be placed) and the press the "submit" button. After this step go to the console and type "8".

NOTE: To choose another route simply press CTRL+C on the console and launch it again (roslaunch mission_planning mission_planning.launch) .

If you get and error on step 3 which has to do with the GPS, it happend because the GPS was not found at the port specified in the launchfile. To solve this go to `workingcopies/lar5/src/sensors/novatel_span/novatel-master/launch` and edit novatel_for_psr.lauch.

Look out for param name="port" value="/dev/ttyUSB3" and change to USB port to the one that the GPS is connected to.

You can change the publish rate of the GPS coordinates by editing the novatel_for_psr.lauch file and changing the value of the GPS default log period. param name="gps_default_logs_period" value="0.25"

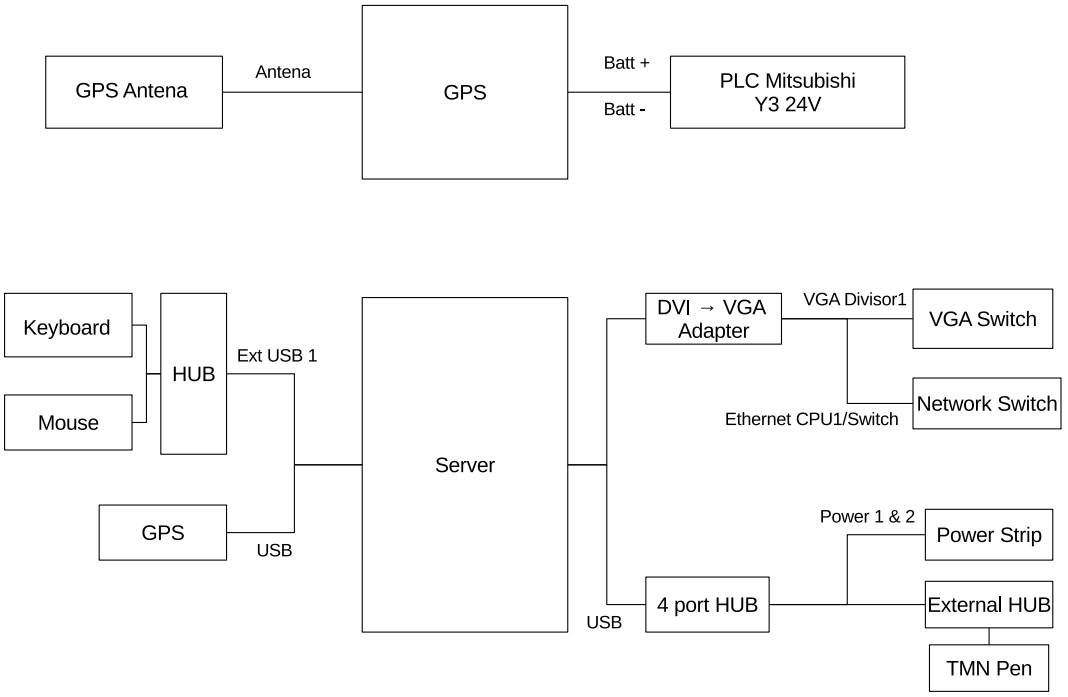
A.7 How to install the novatel package

This is only necessary if, for any reason all data stored in the LAR server is lost. Or another operative system that is not based on linux is chosen.

To install this package follow the instructions on GAVLab official Git webpage `https://github.com/GAVLab/novatel` (visited on 09/09/2015)

Appendix B

Connection Diagram


GPS Antena
Antena
GPS
Batt +
Batt -
PLC Mitsubishi
Y3 24V
Keyboard
Mouse
HUB
Ext USB 1
GPS
USB
Server
DVI → VGA
Adapter
VGA Divisor1
VGA Switch
Network Switch
Ethernet CPU1/Switch
Power 1 & 2
Power Strip
4 port HUB
External HUB
USB
TMN Pen

Appendix C

Algorithm to decode Google Polyline

```
vector<double>Decompress(string encodedPoints, int precision){

    int len = encodedPoints.length();
    int index =0;
    double lat =0;
    double lng = 0;
    vector<double> array;

    while (index < len) {

        int b;
        int shift =0;
        int result =0;
        do {
            b = (int)encodedPoints[index++] - 63; //gets ascii value of the
     char
            //result |= (b & 0x1f) << shift;
            result |= (b & 31) << shift;
            shift +=5;
        } while (b>=32);
        int dlat= ((result & 1) ? ~(result >> 1) : (result >> 1));
        lat+=dlat;
        shift =0;
        result =0;
        do {
            b = (int)encodedPoints[index++]- 63; //gets ascii value of the
    char
            result |= (b & 31) << shift;
            shift +=5;
        } while (b>=32);
        int dlng = ((result & 1) ? ~(result >> 1) : (result >> 1));
        lng+=dlng;

        array.push_back(lat*pow(10,-precision));
        array.push_back(lng*pow(10,-precision));
    }
    return array;
}
```